Simulation of Spray Deposition in Adults Nasal Airway

by

Milad Kiaee Darunkola

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Mechanical Engineering
University of Alberta

Abstract

The goal of this thesis work was to develop an idealized adult nasal airway geometry capably of mimicking average regional nasal deposition of droplets emitted from pharmaceutical nasal sprays. The first part of this thesis examined regional deposition within the nose for nasal sprays over a large and wide-ranging parameter space by using numerical simulation. A set of seven realistic adult nasal airway geometries was defined based on Computed Tomography (CT) images. Deposition in six regions of each nasal airway geometry (the vestibule, valve, anterior turbinate, posterior turbinate, olfactory, and nasopharynx) was determined for varying particle diameter, spray cone angle, spray release direction, particle injection speed, and particle injection location. Penetration of nasal spray particles through the airway geometries represented unintended lung exposure. Penetration was found to be relatively insensitive to injection velocity, but highly sensitive to particle size. Penetration remained at or above 30% for particles exceeding 10 microns in diameter for several airway geometries studied. Deposition in the turbinates, viewed as desirable for both local and systemic nasal drug delivery, was on average maximized for particles in the range ~20-30 microns in diameter, and for low to zero injection velocity. Similar values of particle diameter and injection velocity were found to maximize deposition in the olfactory region, a potential target for nose-to-brain drug delivery. However, olfactory deposition was highly variable between airway geometries, with maximum olfactory deposition ranging over two orders of magnitude between geometries. This variability is an obstacle to overcome if consistent dosing between subjects is to be achieved for nose-to-brain drug delivery.

These simulation results were then used to establish target values of regional deposition for the idealized geometry. Characteristic geometric features observed to be common to all the realistic

nasal airway geometries studied were extracted and included in the idealized geometry. Additional geometric features and size scaling were explored at various stages of the project, in order to enhance deposition in specific regions based on the results of simulations done in earlier versions of the geometry. In total, more than hundred thousand of simulation cases were conducted across a range of particle parameters and geometric shapes in order to reach the final idealized geometry presented herein. The proposed idealized geometry has potential use in the development and testing of nasal drug delivery systems, allowing researchers to estimate *in vivo* regional nasal deposition patterns using a simple benchtop test apparatus.

Preface

This thesis consists of two main parts (chapter 2&3) and is accomplished by me, Milad Kiaee Darunkola. The style of the thesis is in manuscript format.

The second chapter has been published as a journal paper in which I am the primary author in a peer reviewed publication. It is published as Milad Kiaee, Herbert Wachtel, Michelle L Noga, Andrew R Martin, Warren H Finlay, 2018. “Regional deposition of nasal sprays in adults: A wide ranging computational study,” International Journal for Numerical Methods in Biomedical Engineering, volume 34 issue 5. I was responsible for modeling, simulation and visualization. I also prepared the related materials to be written. Dr. Michelle Noga provided the realistic CT scan geometries. Anonymized CT scans were acquired retrospectively from patients scanned for clinical purposed at the University of Alberta Hospital, with Health Research Ethics Board approval. The CT Scan reconstruction was accomplished by John Chen and readied for 3D printing by Eric Bracke. Drs. Warren Finlay and Andrew Martin assisted with writing the manuscript. Moreover, some of the text from Chapter 1 is taken from this published paper.

The third part of thesis is going to be submitted for publication in Journal of Biomechanics. I was responsible for modelling, simulation and visualization. Luba Slabyj, Drs. Warren H Finlay and Andrew R Martin assisted me with writing the manuscript.

Dedicated to my beloved Hye Rin

Acknowledgment

This thesis would have not been possible if it was not due to the continuous support of my PhD supervisors.

Dr. Professor Warren H. Finlay with his wisdom and insight was very helpful. His extraordinary background provided crucial guidance while his modest attitude provided the space for me to be creative. Furthermore, he was perfectly supportive of my research facility demands. Dr. Finlay's vast knowledge in the field of respiratory drug delivery and his solid reasoning skills were the main support of this thesis. Working under Dr. Finlay will always remain as an honor in my resume. Dr. Andrew R. Martin was my second supervisor. His brilliant suggestions initiated many building blocks of the thesis. His professional attitude, precession and kindness motivated me during the hardest challenges of my PhD studies. The high technical and ethical level I witnessed in Dr. Martin, both as superior and instructor, is a milestone for my future career. I am grateful to Dr. Alexandra Kormakova to accept to be part of my supervisory committee. Her questions and suggestions during my candidacy exam were very helpful.

I would also like to thank the aerosol lab technician: Helena Orszanska and my lab mates: Scott Tavernini, Conor Rusycki, Alvin Ly, John Chen and Tyler Paxman. They were all capable, friendly and cooperative.

Last but not least, I would like to thank the University of Alberta and the Department of Mechanical Engineering for providing the opportunity of PhD studies since January 2015. The peaceful environment, friendly staff and useful facilities made my studies more pleasant.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Nasal drug delivery is widespread in the treatment of allergic rhinitis (Keith et al. 2012; Bousquet et al. 2008). Local delivery of corticosteroids to the nasal airways by means of nasal spray pumps is a mainstay for treatment of allergic rhinitis symptoms. In addition, several classes of marketed products have been developed for systemic drug delivery through the nose. Rapid and direct absorption of drug through the nasal epithelium to the systemic circulation enables fast onset of action; fittingly, marketed products in this category include those intended to treat migraine headaches (Tepper 2013) and break-through cancer pain (Taylor et al. 2014). Finally, intranasal drug delivery has received considerable recent attention as a route of administration through which to target the brain (Pardeshi and Belgamwar 2013; Bahadur and Pathak 2012), and thus treat central nervous system diseases such as Alzheimer's and Parkinson's.

For all these applications, a critical consideration is the deposition pattern of nasal spray droplets or aerosols within the nasal airways. Droplets collected in the anterior nasal passages may pool and drip from the nostrils (Chet L. Leach et al. 2015), whereas droplets passing through the nasal cavity to the nasopharynx and larynx miss their site of action or absorption and cause an unpleasant taste upon deposition (Chet L. Leach et al. 2015; Djupesland and Skretting 2012) or may penetrate further to the lungs where toxicological implications must be considered (Djupesland et al. 2004; Suman et al. 1999). In the case of nose-to-brain delivery, the distribution of deposited spray droplets over the nasal epithelium is particularly critical. While the olfactory region of the nasal mucous membrane offers a potential pathway to the brain (Lehrer 2014), it represents only a small fraction (~5-10%) of the total human nasal mucosal surface. Drug delivered to the remaining ~90-

95% of the nasal mucosal surface will be absorbed to the bloodstream, or removed by clearance mechanisms, and hence not directly available to the brain.

In vivo data describing regional deposition of nasal sprays assessed using gamma scintigraphy is available for a limited number of devices and formulations, e.g. (Chet L. Leach et al. 2015; Al-Ghananeem et al. 2008). Unfortunately, the cost and time requirements associated with conducting *in vivo* studies are such that these studies are rarely conducted in the early stages of nasal drug product development, where they would provide valuable feedback to developers. *In vitro* techniques using anatomically sectioned nasal airway replicas have been explored as a means to predict *in vivo* deposition patterns (Xi et al. 2017; Hughes et al. 2008). However, the range of parameters that may affect regional deposition is wide, such that a number of researchers have turned to in *silico* numerical simulation methods to investigate variation in regional deposition that arises as a function of, e.g., droplet size, initial droplet velocity, spray cone angle, spray cone direction, inhalation flow rate, nozzle insertion depth, and nasal airway geometry (Rygg et al. 2016; Schroeter et al. 2006).

For aerosol drug delivery to the lungs, various researchers have described *in vitro* methods using realistic or idealized airway geometries selected to mimic average deposition measured in *in vivo* studies (Below et al. 2013; Javaheri et al. 2013; Delvadia et al. 2012; Golshahi and Finlay 2012; Longest et al. 2012; Byron et al. 2010). Such geometries can function as a reference for *in vitro* experiments or *in silico* simulations, facilitating prediction of *in vivo* performance at early stages of drug or device development, and allowing comparable results to be obtained between laboratories. For nasal drug delivery, a similar geometry mimicking *in vivo* regional spray deposition in an average sense has not been established.

A previous attempt to develop an idealized nasal airway geometry (Liu et al. 2009) used a combination of computational fluid dynamics, cross sectional averaging and two dimensional image processing. Such an approach may be useful for particle penetration to lung in which the total deposition shows a linear or nearly linear behavior (as will be shown in Chapter 2 of this thesis). However, the regional deposition of particles or droplets within the nose is inherently a nonlinear function of the shape of the geometry, making a linear superposition inherently inaccurate. In other words, there is no evidence that an idealized geometry based on linear averaging of realistic geometries would produce the average of deposition in those geometries. Furthermore, an idealized geometry based on cross-sectional averages could prove complicated and lead to manufacturing problems.

For inhalation drug delivery to infants, which occurs through the nasal airways, an alternative approach has previously been taken to develop an idealized infant nasal geometry. This approach focused on geometric pattern extraction in an heuristic manner (Javaheri et al. 2013; Golshahi and Finlay 2012). The approach resulted in a significantly simpler and smoother geometry that is also easier to manufacture. Although the qualitative approach toward feature extraction in these studies is less mathematically rigorous than that adopted by (Liu et al. 2009), it favors the important concept of nonlinear structures.

The current thesis work was undertaken with the goal of developing an idealized nasal airway geometry that mimics regional nasal deposition of nasal spray droplets in adult subjects. This was accomplished in two stages. First, as presented in Chapter 2, regional deposition within the nose was examined using numerical simulation over a large and wide-ranging parameter space. A set of seven realistic adult nasal airway geometries was defined based on Computed Tomography

(CT) images of adult subjects. Deposition in six regions of each nasal airway geometry (the vestibule, valve, anterior turbinate, posterior turbinate, olfactory, and nasopharynx) was determined for varying particle diameter, spray cone angle, spray release direction, particle injection speed, and particle injection location.

These simulation results were then used to establish target values of regional deposition for the idealized geometry. As described in Chapter 3 of this thesis, characteristic geometric features observed to be common to the realistic nasal airway geometries studied were extracted and included in the idealized geometry. Additional geometric features and size scaling were explored at various stages of the project, in order to enhance deposition in specific regions based on simulations in earlier versions of the geometry. In total, more than hundred thousands of simulation cases were conducted across a range of particle parameters and geometric shapes in order to reach the final idealized geometry presented in Chapter 3. The potential impact of the geometry in the development and testing of nasal drug delivery systems is discussed in Chapter 4, where possible direction for future work are also described.

Chapter 2: Regional Deposition of Nasal Spray in Adults: A Wide Ranging Computational Study

2.1 Introduction

The present work was conducted to build upon previous *in silico* studies by implementing a large-scale simulation set, so as to simulate regional deposition of nasal sprays over a large parameter space. A set of realistic adult nasal airway geometries from seven subjects was defined based on Computed Tomography (CT) images. Deposition in six regions of each nasal airway geometry (the vestibule, valve, anterior turbinate, posterior turbinate, olfactory, and nasopharynx) was determined for varying particle diameter, spray cone angle, spray release direction, particle injection speed, and particle injection location. Particular attention was paid to parameter combinations that maximized olfactory deposition, given the low olfactory deposition fractions simulated in previous studies (Keeler et al. 2015; Schroeter et al. 2006).

2.2 Materials and Methods

2.2.1 Airway Geometries

CT images of the nasal airways from the nares to below the larynx were obtained for seven adult subjects averaging 60 years old (see Table 2.1).

Anonymized CT scans were acquired retrospectively from patients scanned for clinical purposes at the University of Alberta Hospital, with Health Research Ethics Board approval. In addition to being assessed as normal at the time of scanning, their nasal airways were confirmed to be normal by a radiologist reviewing the CT images. CT imaging was performed on either a Siemens Somatom Flash or Definition scanner, with a reconstructed slice thickness of one millimeter and in plane resolution of 0.035 to 0.039 mm.

Table 2.1 Relevant information for the 7 subjects. See Figure 2.3 for approximate locations of the different listed airway regions (Vestibule, Valve, Anterior Turbinates, Posterior Turbinates, Olfactory and Nasopharynx).

Sub No	Sex	Age (years)	Airway Surface Area (cm^2)							Volume (cm^3)
			Total Area (cm^2)	Vesti	Valve	Anterior	Posterior	Olf	Naso	
1	M	60	337.6	12.6	20.6	36.7	171.8	10.0	85.8	59.6
2	F	50	315.5	10.1	17.2	22.3	154.5	6.1	106.0	73.1
3	M	57	320.2	14.5	18.1	22.8	192.0	7.4	90.0	59.2
4	M	54	344.7	11.8	24.6	19.7	161.3	8.8	116.0	71.5
6	F	72	317.8	14.3	32.0	53.6	137.7	8.6	69.3	59.0
7	M	62	308.2	14.3	30.7	27.0	140.8	12.0	84.2	56.6
8	M	63	323.6	14.2	20.8	31.3	163.0	10.4	81.8	61.8

The DICOM files from the CT images were processed using ScanIP (Simpleware, UK), which involved removal of the sinuses and segmentation to define the nasal airways proximal to the upper trachea. The segmented airways included the laryngeal region. The resulting airway surfaces were smoothed locally using Meshmixer (Autodesk, USA), followed by iterative global smoothing with 3-maticSTL (Materialise, UK). The ratio of volume to surface area was recorded after each smoothing iteration, and smoothing was stopped once this ratio converged to 2 decimal places. Topological flaws (e.g. excessively high aspect ratio, missing triangles, excessive node density, self-intersections) in the reconstructed STL files associated with each subject's nasal airways were repaired using Netfabb (Autodesk, USA) and MeshLab (Visual Computing Laboratory, Italy), visualised with VTK C++ and Paraview (Kitware, USA). This required a number of manual manipulations including closing holes, stitching triangles, fixing flipped triangles, removing double triangles and degenerate faces. Views of the final nasal airway walls for the seven subjects are shown in Figure 2.1.

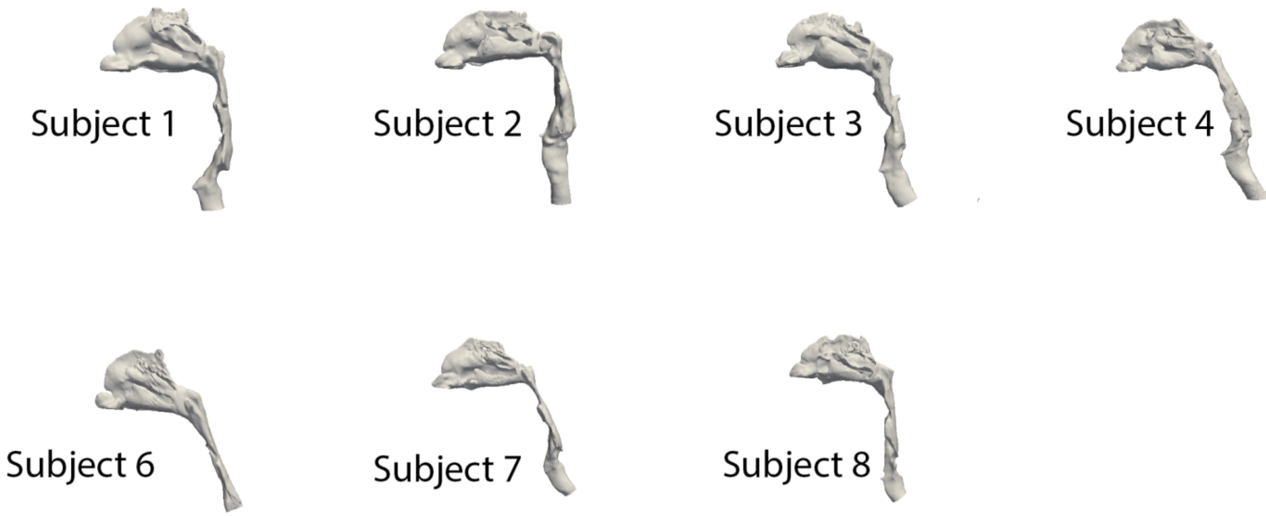


Figure 2.1 Perspective views of the model airways used in this study. Subjects 5, 9 and 10 are excluded due to geometric defects observed during meshing.

2.2.2 Computational Fluid Dynamics of Airflow

The fluid dynamics in each of the subject's nasal airways was simulated by solving the incompressible laminar and steady state Navier-Stokes equations using OpenFOAM version 3.0.1 (OpenFOAM Foundation Ltd, UK). OpenFOAM solves a discretized approximation to the Navier-Stokes equations using a finite volume method. The STL file for each subject was imported into OpenFOAM's SnappyHexMesh routine to produce a mesh of hexahedral elements upon which numerical solution to the Navier-Stokes equations was performed. The mesh generation tool includes refined grid spacing in boundary regions close to the walls. Although CT images were obtained for 12 subjects, geometric and topological defects in the geometries produced by the reconstruction software for subjects 5 and 9-12 were severe enough in those subjects' airways that the segmentation and meshing software did not produce a geometry and mesh of sufficient quality to proceed with a CFD solution.

For the remaining 7 subjects listed in Table 2.1, a CFD solution was obtained using OpenFOAM's PIMPLE solver. PIMPLE uses SIMPLE (Semi-Implicit Method for Pressure linked Equations) during the inner linear solver iterations and PISO (Pressure Implicit Splitting of Operator) during the nonlinear outer iterations. Spatial discretization was second order ("Gauss linear" in OpenFOAM, with cell limiting applied to the gradient terms). Grid convergence studies were performed to determine the number of cells required to achieve grid independence (within 10%) in the value of pressure drop through the airways of each subject. The number of cells was thus subject dependent but ranged from 600,000 (for subject 3) up to 3,600,000 (for subjects 4 and 6).

In order to mimic delivery of sprays delivered through a single nostril, a zero-velocity boundary condition was set at the entrance of one nostril. At the entrance to the other nostril, the flow rate was set at 15 l/min, in keeping with an assumption of laminar flow (Tu et al. 2013). For the 15 l/min nostril, a parallel, uniform flow velocity field boundary condition was used. At the exit, a Neumann condition was used for velocity and pressure, coupled with the mass flow rate specified by the inlet velocity field.

The fluid flow was simulated separately for 15 l/min flow through the left nostril, and then another simulation was performed for 15 l/min flow through the right nostril. The final set of fluid flow simulations thus consisted of 14 individual CFD simulations.

2.2.3 Measurements of Pressure Drop

To provide partial validation of the CFD simulations, a physical replica of each subject's nasal airways was built from plastic (Objet VeroGray RGD850; Stratsys, Ltd.; Eden Prairie, MN, USA) using a PolyJet 3D printer (Objet Eden 350V High Resolution 3D Printer; Stratsys, Ltd.; Eden Prairie, MN, USA) as described recently (Chen et al. 2017). The pressure drops across the nasal airways, from the entrance of the nares to an outlet within the trachea below the larynx, was measured for all seven subjects using a digital manometer (HHP-103, Omega, Canada) for flow rates ranging from 10-90 litres/minute, measured with a TSI 4000 flow meter (TSI, USA).

2.2.4 Lagrangian Particle Tracking

Particles were injected over a variety of positions and velocities within the entrance region of the nares. These particles were assumed to be stable (i.e. non-evaporating) with no bounce (i.e. they stick) upon deposition with an airway surface. A particle density of 1000 kg/m^3 was assumed. Particle trajectories and their deposition locations were then calculated by solving Newton's second law for each particle using OpenFOAM's IcoUncoupledKinematicParcelFoam solver. This solver assumes one-way momentum coupling between the particles and the fluid. It was assumed that the only forces acting on the particles are gravity and fluid drag, the latter specified by the Schiller-Neumann drag coefficient:

$$C_D = \frac{24}{Re_P}(1 + 0.17\, Re_P^{0.66}) \qquad (2.1)$$

where Re_p is the particle Reynolds number based on its velocity relative to the fluid velocity, particle diameter, and a kinematic viscosity of air $\nu = 1.5 \times 10^{-5} m^2/s$.

The previously calculated fluid velocity field was interpolated to particle positions using second order interpolation via OpenFOAM's Mean Value Coordinate (MVC) method. Particle positions were advanced in time using a first order implicit Euler method. Grid convergence studies (both in space and time) were performed with respect to the value of regional deposition to determine grid resolution and time step.

In addition, convergence studies were performed to determine the number of particles needed. Particles were injected within the nostril from a planar disk region with 1mm diameter; the position of this disk was varied within the nares using Grasshopper (Rhinoceros, USA) to define 200 random positions for each subject and nostril side (left or right). The injection location was varied from a little inside the entrance of the nares to a little after the entrance of the nasal valve region, with these insertion depths varying approximately in the range of 0.2 to 1.5 cm from the inlet. Figure 2.2 shows the central positions of the injection cones for one nostril of one subject.

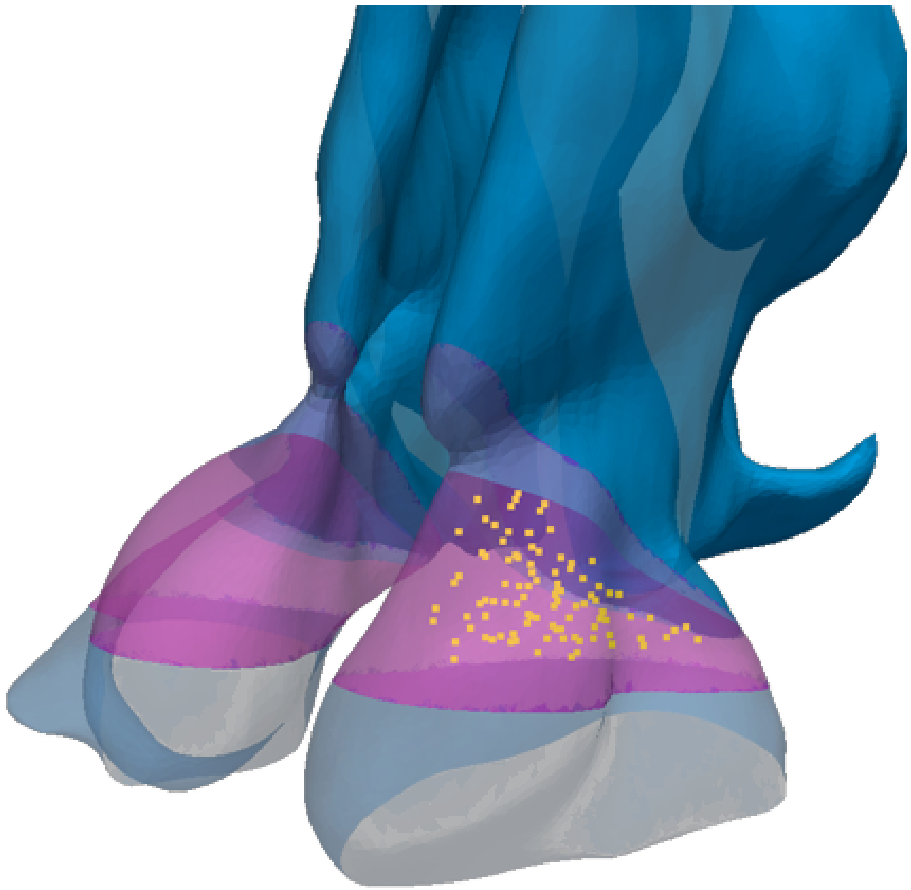


Figure 2.2 The yellow points indicate the center of the spray injection release disk in one nostril of one subject. The shaded purple region shows the approximate defined volume within which these injection locations were randomly placed. For each subject, 200 such injection locations were simulated in each nostril (i.e. 400 points total between the right and left nostrils).

For each disk position, 10,000 particles were injected within the disk. Particle injection velocities were specified to give a cone shape to the injection plume with specified half-angle.

After examining literature values and published data for commercial nasal spray devices, as well as a small number of preliminary simulations over a wide range of parameter values, a subset of parameters and their values were chosen as being most relevant and applicable. Table 2.2 shows these parameters and the range of values for which simulations were performed in each of the seven subjects.

Table 2.2 Parameter values for particle tracking simulations performed in all seven subjects.

Parameter	**Number of Parameter Values Simulated**	**Range of Values**
Particle diameter	5	5 – 40 microns
Spray half cone angle	2	17.5 and 30 degrees from spray cone direction
Spray cone direction	2	Upward (i.e. vertical) and semi-upward (aimed at the nasal valve entrance, approximately 75^0 from vertical)
Particle injection velocity	4	0-20 m/s
Position of injection disk	200	Generated randomly within a defined boundary
Nasal airway geometries	7	Normal airway geometries derived from CT scans (see Table 2.1)
Spray Injection Side	2	Left and right nostrils

Despite having narrowed the parameter space to what was believed to be the most relevant subspace, the parameter ranges in Table 2.2 still required performing a total of 224,000 simulations associated with each individual parameter value. Because of the large computational time of these simulations, they were done in parallel on a computing cluster (SGI Altix XE, 400 nodes, 4160 cores, Compute Canada). To allow assessment of regional deposition, the nasal airway walls were divided into the following regions in each subject: vestibule, valve, anterior turbinate, posterior turbinate, olfactory and nasopharynx. Figure 2.3 shows these regions as defined for one of the subjects. Table 2.1 gives the surface area of these regions for each subject. The regions were defined following the common approach in previous studies (Schroeter et al. 2006) . Furthermore, the criteria for defining regions was approved by an expert radiologist.

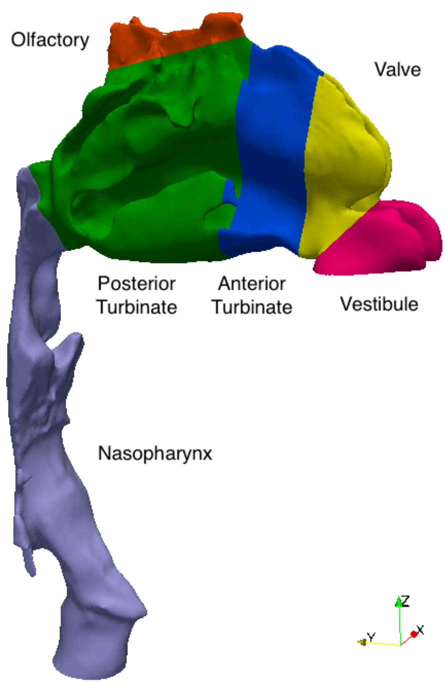


Figure 2.3 The six anatomical regions of the nose as defined in one of the subjects.

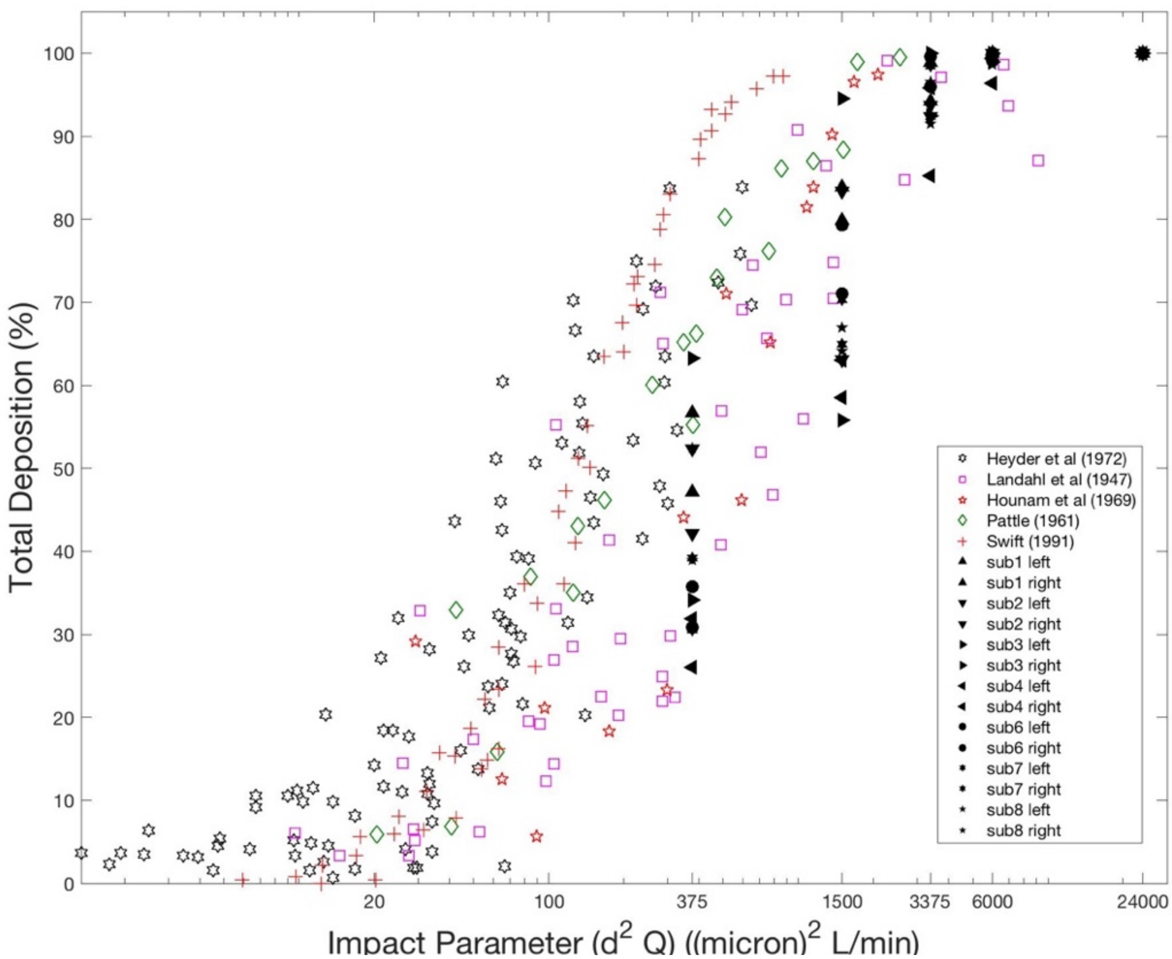
2.3 Results

2.3.1 Validation

Calculated pressure drops in the seven subjects were found to be within 12.5% of the values measured experimentally at 15 l/min in physical replicas of these same subjects' nasal airways (average $\pm$standard deviation in $\Delta p = 29.6 \pm 10.4 Pa$ measured *vs*. $\Delta p = 25.9 \pm 9.7 Pa$ calculated). Total deposition calculated in the nose of the present subjects is shown in Figure 2.4 with particle injection velocity set to zero and a flow rate of 15 l/min.

Similar data from various *in vivo* studies is also shown. Given the well-known large intersubject variability, reasonable agreement with *in vivo* data is seen between calculated total nasal deposition in our seven subjects and total deposition measured *in vivo* in other subjects.

Figure 2.4 Total deposition versus impaction parameter from our CFD particle tracking simulations (solid black symbols) in our seven subjects with 15 L/min through a single nostril is shown along with data from previous in vivo studies in different subjects.



2.3.2 *Regional Deposition*

Because of the sheer volume of the data, it is difficult to summarily present results from the nearly ¼ million individual runs performed. However, Figures 2.5-2.8 present regional deposition data averaged over all particle injections points (which are randomly distributed like that shown in Figure 2.3) and averaged over both cone angles given in Table 2.2, but with only a vertically upward injection considered (as the results for other injection directions did not show interesting or unexpected differences) is shown for injection occurring separately in the left and right nostrils of each subject.

Figure 2.5 shows deposition in the vestibule and the valve regions combined, which unsurprisingly is seen to be highest for the largest particles injected at the highest speeds.

Figure 2.6 combines both anterior and posterior turbinate deposition and is seen to peak in most subjects at a middling particle size and drops off at the higher particle injection speeds.

Figure 2.7 shows the fraction of particles exiting the simulation via the outlet distal to the larynx; these are particles that penetrate the nasal region and enter the lungs. Particle injection speed is seen to have little effect on the fraction of particles penetrating the nose; for all subjects, maximal nasal penetration occurs for the smallest value of particle diameter considered by us (i.e. 5 micrometers). Figure 2.8 shows olfactory deposition in each of the subjects.

Figure 2.8 only shows deposition for upward injection, since this injection direction consistently gave somewhat higher olfactory deposition than semi-upward injection (overall average 1.8% olfactory deposition for upward vs 1.4% for semi-upward for the range of parameter values in Table 2.2).

While Figure 2.8 shows olfactory deposition averaged over all 200 injection locations in each nostril, examination of the data for the individual injection locations shows that some injection locations have much higher olfactory deposition than others.

In particular, there is a distinct region within the vestibule that was found to give considerably higher olfactory deposition. This region is located close to the upper wall of the vestibule region and is highlighted in Figure 2.9 for two of the subjects. Figure 2.9 shows the maximum olfactory deposition that occurs for each different injection point maximized over all the other parameters in Table 2.2. Similar results were seen in all subjects.

Table 2.3 gives absolute maximum values of olfactory deposition occurring in each subject, maximized over all parameter values in Table 2.2 (including injection location). This table shows that with a narrow, subject-specific choice of parameter values it is sometimes possible to have high values of olfactory deposition, despite overall average olfactory deposition being 1.6% for the parameter value range of Table 2.2.

While 100% olfactory deposition is achieved in a few runs, it should be noted that this results from a rare (one out of thousands) combination of the parameter values in Table 2.2 e.g. a very specific release position and initial velocity. Such a precise combination of parameter values is likely not practical to achieve *in vivo*.

Figure 2.5 Deposition in the vestibule and valve regions (combined) of each subject is shown with injection in an upward direction via the left and right nostrils shown separately. The y-axis is particle diameter in micrometers. The x-axis is particle injection velocity in m/s. The data is averaged over 200 injection locations (see Table 2.2 and Figure 2.3) and two cone angles (see Table 2.2).

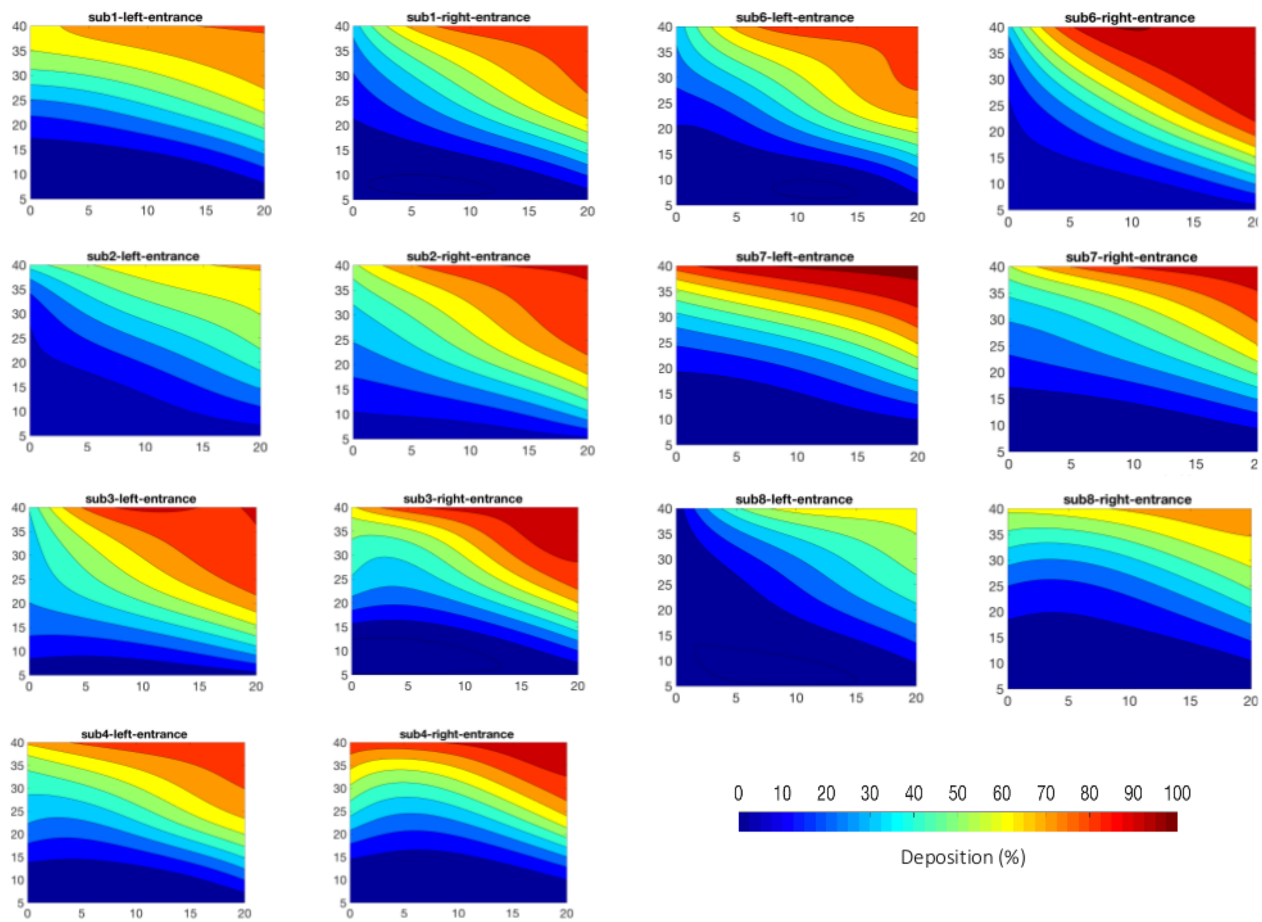


Figure 2.6 Deposition in the turbinates (both anterior and posterior) of each subject is shown with injection in an upward direction via the left and right nostrils shown separately. The y-axis is particle diameter in micrometers. The x-axis is particle injection velocity in m/s. The data is averaged over 200 injection locations (see Table 2.2 and Figure 2.3) and two cone angles (see Table 2.2).

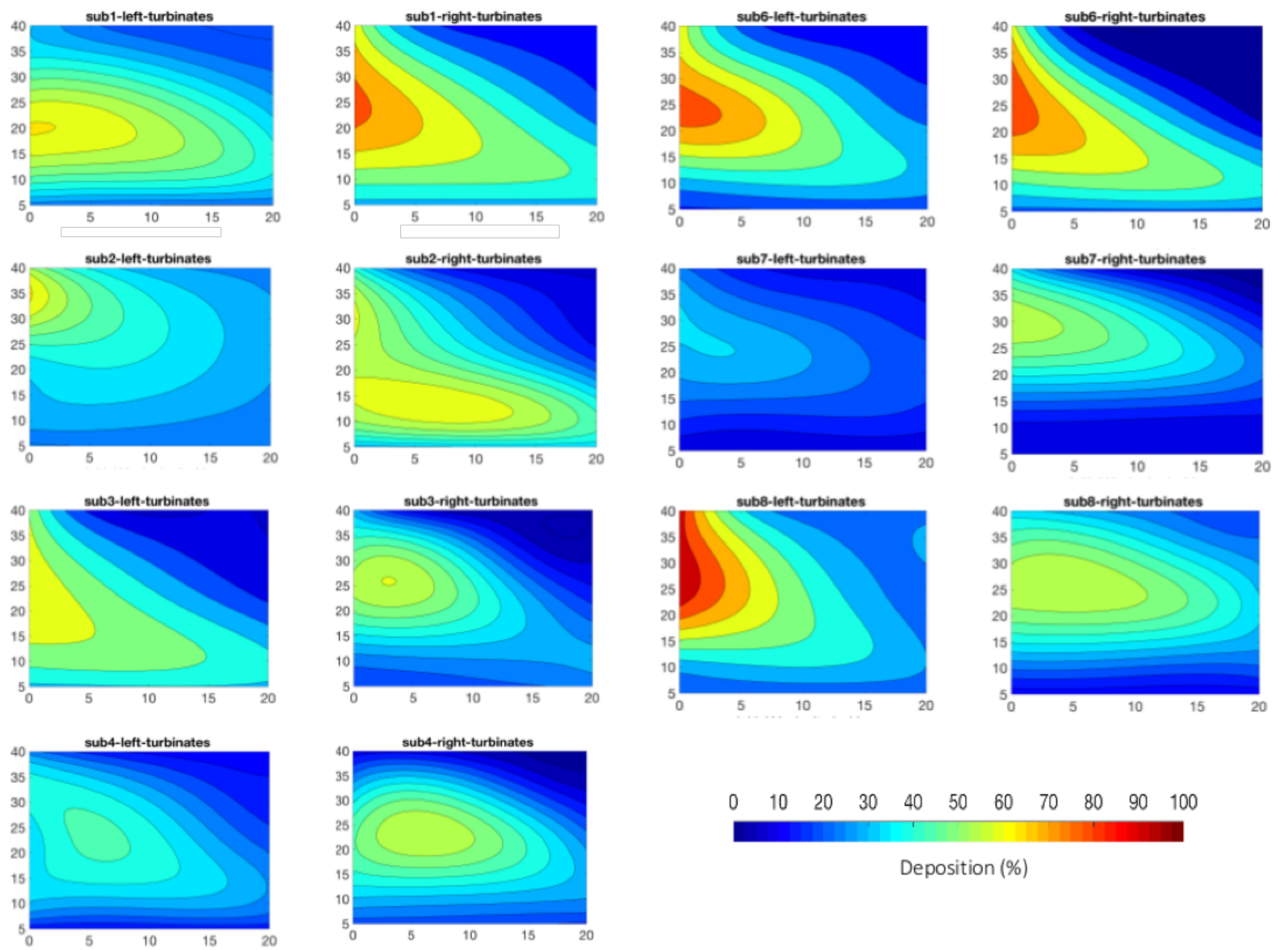


Figure 2.7 Percentage of the particles entering the lungs (i.e. exiting the simulation outlet distal to the larynx) of each subject is shown with injection in an upward direction via the left and right nostrils shown separately. The y-axis is particle diameter in micrometers. The x-axis is particle injection velocity in m/s. The data is averaged over 200 injection locations (see Table 2 and Figure 2.3) and two cone angles (see Table 2).

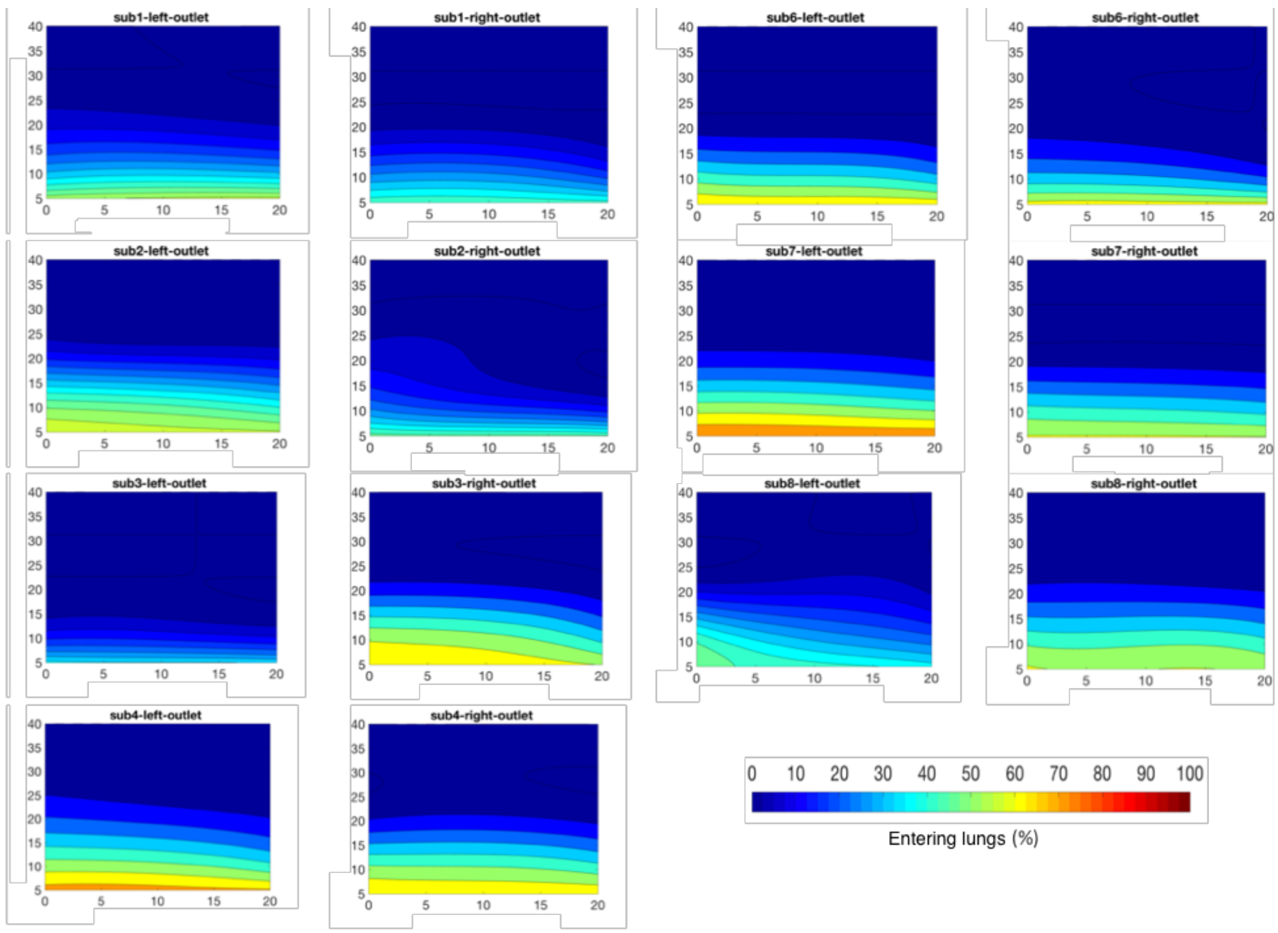


Figure 2.8. Deposition in the olfactory region of each subject is shown with injection in an upward direction via the left and right nostrils shown separately. The y-axis is particle diameter in micrometers. The x-axis is particle injection velocity in m/s. The data is averaged over 200 injection locations (see Table 2.2 and Figure 2.3) and two cone angles (see Table 2.2).

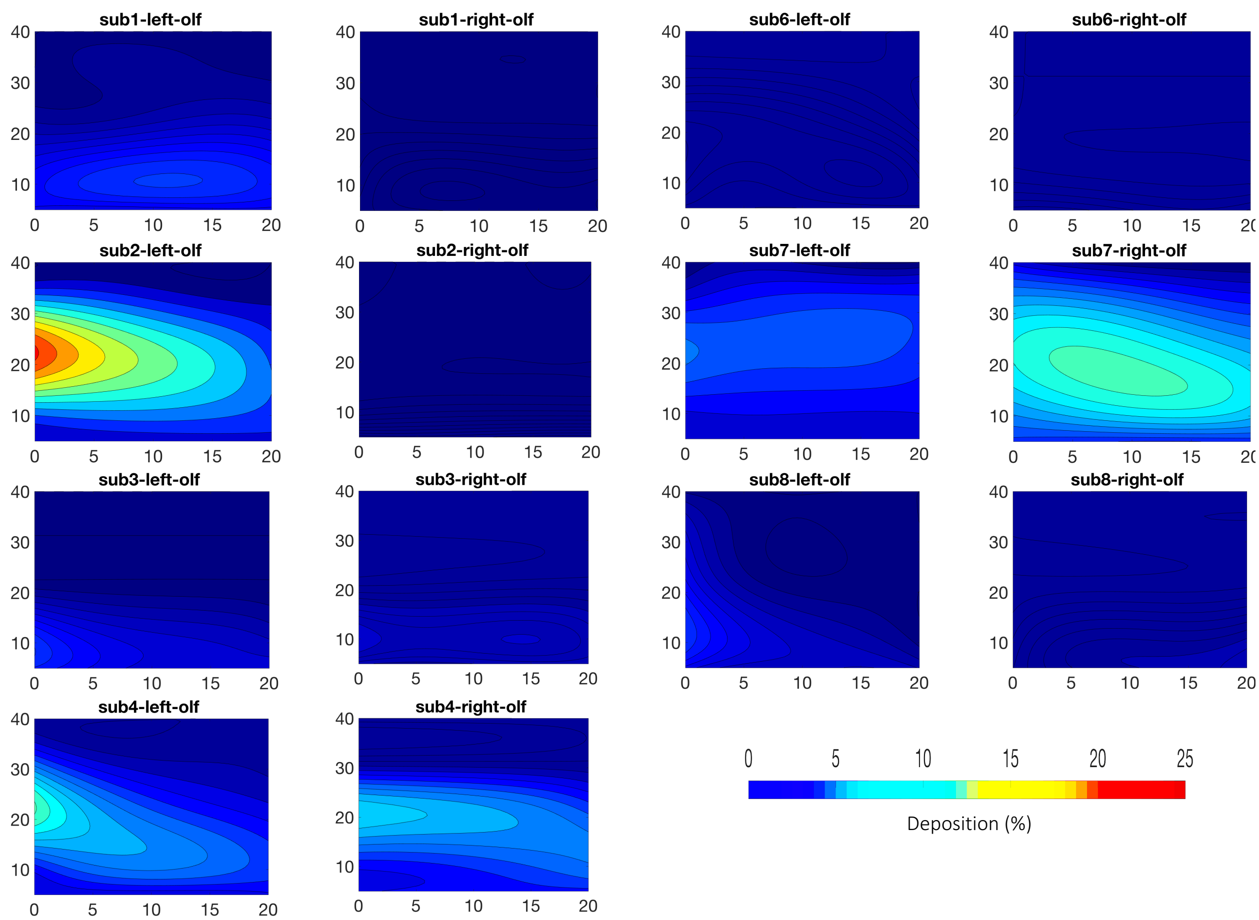
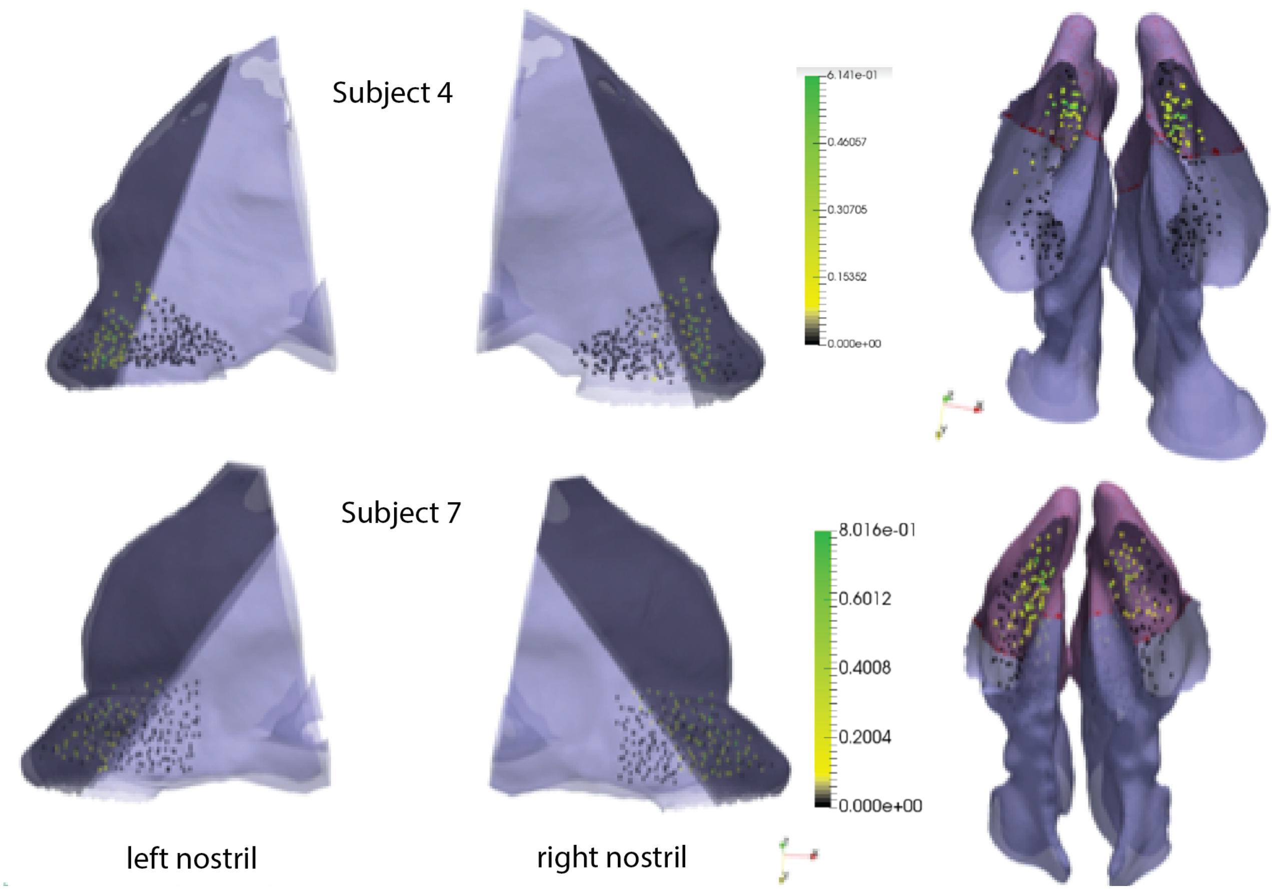


Table 2.3 Global maximum values of olfactory deposition values occurring in each subject for either right or left nostril injection when the injection location and parameter values in Table 2.2 are chosen to maximize olfactory deposition.

Subject	**Nostril**	**Olfactory deposition (%)**
1	Left	38.8
1	Right	4.1
2	Left	98.5
2	Right	6.4
3	Left	88.4
3	Right	53.3
4	Left	100
4	Right	100
6	Left	15.8
6	Right	26.6
7	Left	100
7	Right	100
8	Left	95.4
8	Right	36.9

Figure 2.9 Perspective views of the entrance region (both left and right nostrils are shown) of subjects 4 and 7. The dots show the 200 injection locations in each nostril colored, as denoted in the color bars, by the amount of olfactory deposition occurring when the injection disk is centered at that location (averaged over all other parameter values in Table 2.2). To the left of the color bars the view is side view, while to the right the view is from below the nares. The shaded region indicates the region closer to upper section of the nostril wall. It is observed that the shaded region contains release positions that lead to higher olfactory depositions.



2.4 Discussion

The present study was conducted to explore regional deposition of nasal sprays in the airways of the nose and throat across a wide-ranging parameter space. Numerical simulation was a practical approach to use in conducting such an exploration due primarily to the large number of possible input parameter combinations. In the present work, nearly ¼ million individual simulations were performed. It would clearly not be feasible to conduct the same number of individual experiments using *in vivo* or *in vitro* methods.

A level of confidence in the present results may be gained through comparison with available *in vivo* data describing total nasal deposition of inhaled aerosol particles (Swift 1991; Heyder and Rudolf 1975; Hounam et al. 1971; Pattle 1961; Landahl and Black 1947).

As seen in Figure 2.4, when plotted against the impaction parameter, simulated total deposition data from the present study (for cases with zero particle injection velocity relative to the inspiratory flow) broadly overlap previously reported *in vivo* data. We note that the simulation data is restricted to the upper range of impaction parameter spanned by the *in vivo* data in Figure 2.4 due to the larger size range of nasal spray droplets (5 to 40 μm in diameter) investigated in the present study as compared to typical particle sizes used in *in vivo* aerosol exposure studies (< 10 μm in diameter). Even so, simulated total deposition in the present nasal airway geometries was well below 100% for a significant number of cases (Figure 2.4). This is non-ideal for nasal sprays, where the intention is to delivery drug locally to the nasal airways.

Particles that penetrate the airways of the nose and throat will enter the conducting airways of the lungs, and unintended lung exposure may occur (Djupesland and Skretting 2012; Suman et al. 1999).

Simulated penetration of particles to the lungs is shown in further detail in Figure 2.7.

Several comments can be made. First, it can be noted in view of Figure 2.7 that the percentage of particles reaching the lungs is relatively insensitive to the injection velocity. Second, the influence of particle size on particle penetration to the lungs in pronounced in all 14 geometries. Combined, these observations suggest that in designing nasal delivery devices to avoid lung exposure, emphasis should be placed on the emitted particle size distribution, with details of velocity distribution of emitted particles of secondary importance. Further, while data reported in Figure 2.7 is in broad agreement with European guidelines to limit the fraction of sub 10 μm particles emitted from nasal drug products (Canada 2006), variability between geometries in the percentage of particles penetrating to the lungs is high. In several cases, penetration remains at or above 30% for particles in the range of 10 to 15 μm in diameter.

In addition to the fraction of particles penetrating to the lungs, regional deposition within the nasal airways is of considerable interest. Droplets that deposit in the vestibule and valve may pool and drip from the nostrils (Chet L. Leach et al. 2015). As the nasal mucosa in these proximal regions is non-ciliated, any particles or droplets that do not drip or rapidly absorb will remain in place (Al-Ghananeem et al. 2008) and are subject to mechanical removal, e.g., by wiping or blowing the nose. In contrast, droplets that reach the nasopharynx and larynx miss their target, are subject to rapid clearance (Al-Ghananeem et al. 2008), and may cause an unpleasant taste upon deposition (Chet L. Leach et al. 2015). Accordingly, the intermediate region consisting primarily of the anterior and posterior turbinates would appear to be a preferential target for both local and systemic nasal drug delivery.

Figure 2.6 displays simulated deposition fractions in the combined anterior and posterior turbinates. For all 14 geometries studied, maximum deposition in the turbinates occurred at intermediate particle size, typically between ~20 and ~30 µm. This result is broadly in agreement with results of simulations reported by (Keeler et al. 2015), where for a constant particle injection velocity of 1 m/s deposition in the turbinates peaked between particle sizes of 25 and 30 µm in the majority of subject geometries studied. In addition, in the present study, maximum deposition occurred with zero injection velocity for 10 of the 14 geometries studied, and in all cases, deposition fell off as injection velocity increased above ~5-10 m/s. It appears therefore that combinations of large particle size and high initial velocity relative to the inspiratory air flow promote deposition by impaction in the vestibule and valve, whereas small particles (below ~10-20 µm depending on the individual geometry) are carried past the turbinates to deposit in the nasopharynx, larynx, or the lungs.

Finally, deposition in the olfactory region is of interest for exploratory nose-to-brain delivery (Xi et al. 2016, 2017; Lehrer 2014; Pardeshi and Belgamwar 2013; Bahadur and Pathak 2012) (Warnken et al. 2016; Djupesland 2013). Previous *in vitro* and *in silico* studies have reported low deposition fractions in the olfactory region for nasal sprays, with maximum olfactory deposition between 3-14% (Xi et al. 2016; Schroeter et al. 2006). The present results reported in Figure 2.8 are reasonably consistent with these past studies, although a broader range was observed, in that maximum olfactory deposition averaged over all injection locations ranged from ~ 0.1% up to ~25%. As was the case for turbinate deposition, maximum olfactory deposition occurred at intermediate particle sizes and was associated with low to zero injection velocity for the majority of geometries.

In addition, previous researchers have proposed that olfactory deposition be further increased by limiting injection of particles to sub-regions of the nasal vestibule, specifically to locations near the upper, or front, wall of the vestibule (Xi et al. 2016; Schroeter et al. 2006). Consistent with these previous observations, Figure 2.9 indicates very low to zero olfactory deposition for particles injected into a region approximating the lower half of the vestibule, but considerably higher olfactory deposition for particles injected into the upper half of the vestibule.

While similar trends were observed in all geometries studied in the present work, the variability in olfactory deposition between geometries is notable. This is a considerable obstacle to overcome if consistent dosing between subjects is to be achieved for nose to brain drug delivery.

2.5 Conclusions

The present numerical simulations were conducted to provide a data set describing regional deposition of nasal sprays over a wide-ranging parameter space. Penetration of nasal spray particles through the airways of the nose and throat was found to be relatively insensitive to injection velocity, but highly sensitive to particle size. Penetration remained at or above 30% for particles exceeding 10 μm in diameter for several airway geometries studied.

Deposition in the turbinates, viewed here as desirable for both local and systemic nasal drug delivery, was on average maximized for particles ranging from ~20-30 μm in diameter, and for low to zero injection velocity. Similar values of particle diameter and injection velocity were found to maximize deposition in the olfactory region, a potential target for nose-to-brain drug delivery. However, olfactory deposition was highly variable between airway geometries, with maximum olfactory deposition averaged over all injection locations ranging over two orders of magnitude between geometries.

Chapter 3: An Idealized Geometry that Mimics Average Spray Deposition in Adult Nasal Airway

3.1 Introduction

As described in the preceding chapter, the adult nasal airway geometry exhibits complex morphology and intersubject variability (Garcia et al. 2009; Churchill et al. 2004). Particle deposition within the nasal airways is an important consideration in the design and evaluation of intranasal drug delivery systems. In particular, the regional deposition pattern of drugs administered intranasally is expected to impact their therapeutic effectiveness. Many aspects of regional deposition are thought to play a role. For instance, droplets collected in the anterior nasal passages may pool and drip from the nostrils. Conversely, droplets passing through the nasal cavity to the throat miss their site of action, and might penetrate on to reach the lungs, where adverse side-effects could occur.

Numerous *in vivo* and *in vitro* experimental studies have been performed in order to measure particle deposition in the nasal airways (Schroeter et al. 2015; Javaheri et al. 2013; Shah et al. 2013; Byron et al. 2010; Liu et al. 2010; Heyder 2004; Hahn et al. 1993; Heyder and Rudolf 1975). Several computational studies have also been performed using realistic nasal airway geometries (Keeler et al. 2015; Patel et al. 2015; Schroeter et al. 2010, 2012, 2015; Wang et al. 2012; Rhee et al. 2011; Weinhold and Mlynski 2004).

Just as therapeutic benefit can be related to the regional deposition pattern in the nasal airway, so too can unwanted side effects or the outright failure of inhaled sprays to achieve their intended effect. As noted above, spray droplets deposited at the beginning of entrance region (i.e. the nasal vestibule) can drip from the nostrils (Chet L Leach et al. 2015). Droplets depositing distal to the turbinate region will either fail to have the desired treatment effects or end up in regions where they are considered potentially harmful (Chet L Leach et al. 2015; Djupesland and Skretting 2012; Djupesland et al. 2004; Suman et al. 1999).

As seen in Chapter 2 of this thesis, many factors can influence regional deposition of nasal sprays. These include, but are not limited to, the distributions of size and velocity of droplets emitted from nasal spray pumps, the spray cone angle, the orientation angle of the spray with respect to the nasal inlet, and the insertion depth of the spray tip into the nostril. Adding to these the numerous geometric features of the nasal airway that can influence regional deposition patterns, the scope of numerical or experimental studies of regional deposition can become very large. For this reason, a reference idealized geometry would be extremely beneficial in reducing the computational or experimental burden, provided that measurements made using that idealized geometry could, with confidence, be expected to anticipate average values in a larger set of nasal airway geometries. Furthermore, the simplicity of such a geometry would make these analyses more feasible. In experiments, fewer small-scale features or extreme convolutions in a given region makes assay and cleaning easier. In simulations, an idealized geometry makes the discretization less complicated and hence the simulation is less expensive[1].

---

[1] This can be seen specifically in faster process and simpler outcome for the computational grid.

The above considerations motivate development of an idealized geometry which mimics the average deposition in a target population of adult subjects with normal (non-pathological) nasal airways. The current study seeks a geometry that mimics the average regional nasal spray deposition observed in the realistic geometries reported in Chapter 2.

3.2 Methods

3.2.1 Idealization of Airway Geometries

As described in chapter 1, ten individual subjects' nasal airway geometries were obtained using a sectional Computational Tomography (CT) scans (Figure 3.1). Segmentation and reconstruction of the cross sections produced three-dimensional surfaces. The subjects ranged in age from 27-72 years old, included 7 males and 3 females, and resulting scans covered from the nares to below the larynx regions. The medical imaging procedure was approved by the Health Research Ethics Board at University of Alberta.

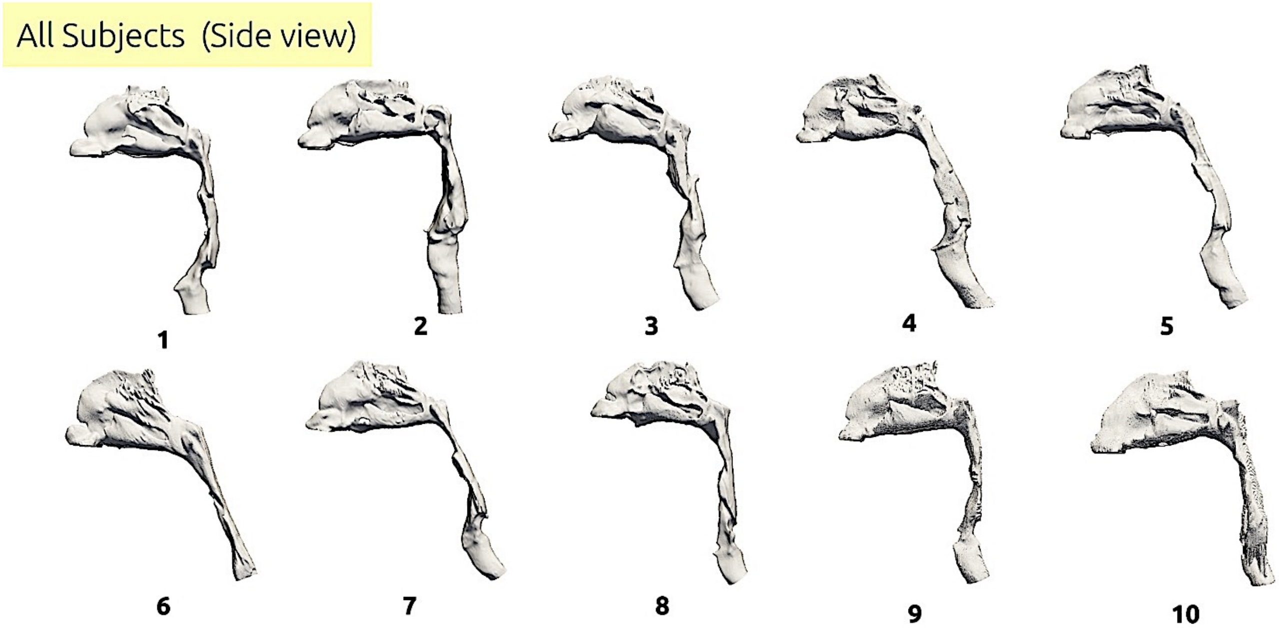


Figure 3.1 Side view of the ten geometries used in this study. CFD results of seven of these geometries (subjects 1, 2, 3, 4, 6, 7 and 8) using both nostrils (one at a time) are given in chapter 1.

We denoted these 10 geometries as realistic geometries. Since the right and left sides of the realistic geometries were nearly independent (Bates et al. 2015; Wen et al. 2007), it was possible to treat them separately as independent case studies, thus essentially doubling the number of reference realistic geometries to twenty. For the purpose of creating an idealized airway, the right and left sides were made symmetrical, consisting of two identical half upper airways, with each half of the airway started from an individual nostril and converging at the beginning of nasopharynx. The two half airways were assumed to be separated proximal to the nasopharynx.

The realistic geometries were all complex, containing numerous features at different scales. From the twenty geometries, fourteen were previously selected for a wide ranging computational parameter space exploration (chapter 1 of thesis). The resultant regional deposition pattern sets a target for evaluating the suitability of the present idealized geometry. The geometry of the remaining six realistic nasal airway realizations was also helpful for the purposes of qualitative observation.

The realistic geometries were available in three dimensions as stereolithography (STL)[2] files, and the global coordinate system was chosen as right handed and Euclidean. The $+y$ direction was defined toward the back of the head and tangent to the inlet surface of the nostril. The $+z$ direction was defined upwardly and normal to the inlet and was called "up".

Slicing the surface geometries on the xz plane resulted in a set of curves containing one or more components, each of which was a simply connected curve. Without loss of generality, we denoted each set as one cross section.

Cross sections were seen to undergo considerable changes in shape when proceeding the y direction. In particular, a shape bifurcation was seen to occur within the turbinates, with the additional branch eventually turnings back to the upper turbinates and creating a semi-circular cross section at the junction between the posterior turbinate and the nasopharynx.

[2] STL files contain information from the triangulated surface geometry. Three vertices and a normal vector from each triangle are stored sequentially in a list. In this study, the STL files contain tens of thousands of triangles.

This second branch contains many small features. The center of area of the cross section produces a nearly flat line in the y axis. However, the bifurcation branch reduces its z components in a nearly linear manner and fades away in the $+y$ direction. The lower part of the cross section rolls upward within the posterior turbinate and becomes further convoluted. The resulting Y-shape cross section vanishes as the nasopharynx is approached. Figure 3.2 shows this so-called Y-shape concha. These observations suggest a special role of the turbinate region in the particle deposition behavior.

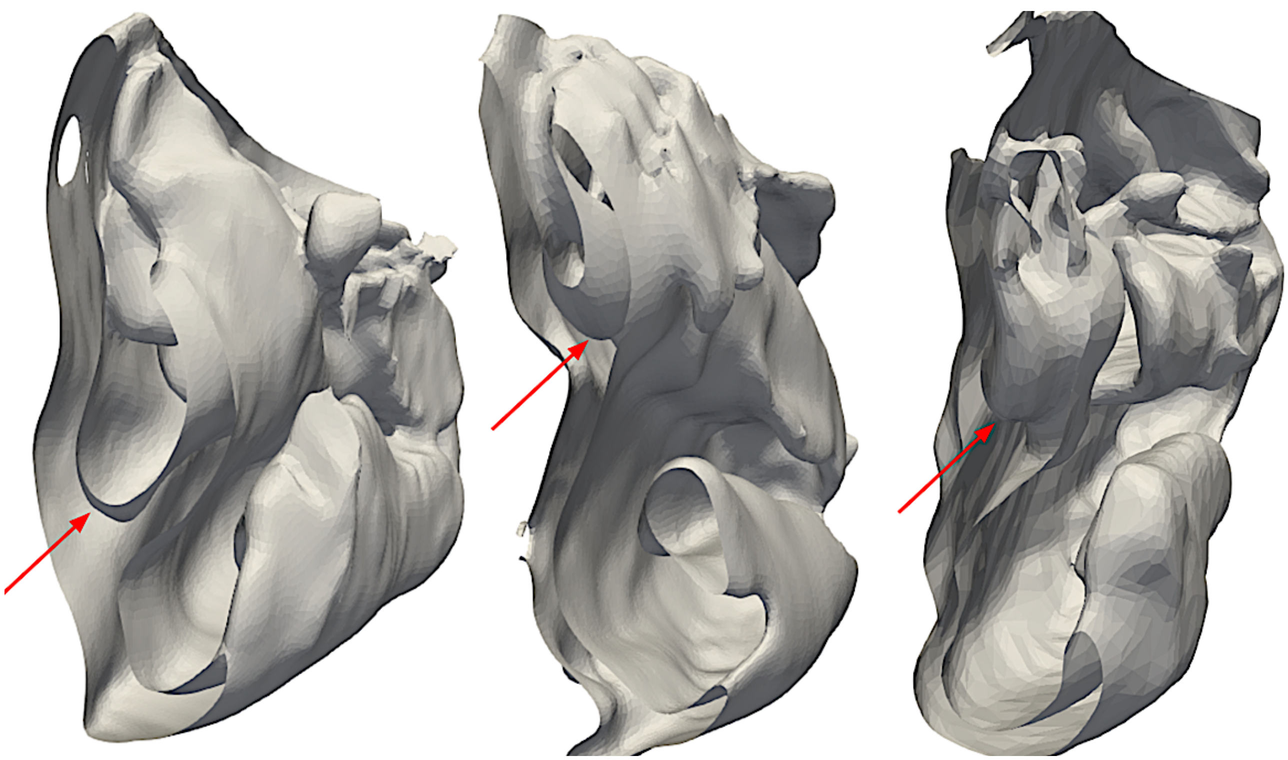


Figure 3.2 Portions of the turbinate region in subjects 1, 2 and 4 are shown (left to right) around the same cut plane. A significant common feature in all geometries is the Y-shaped concha. As is pointed out by the arrow for these examples the position of this Y-shape varies in different subjects.

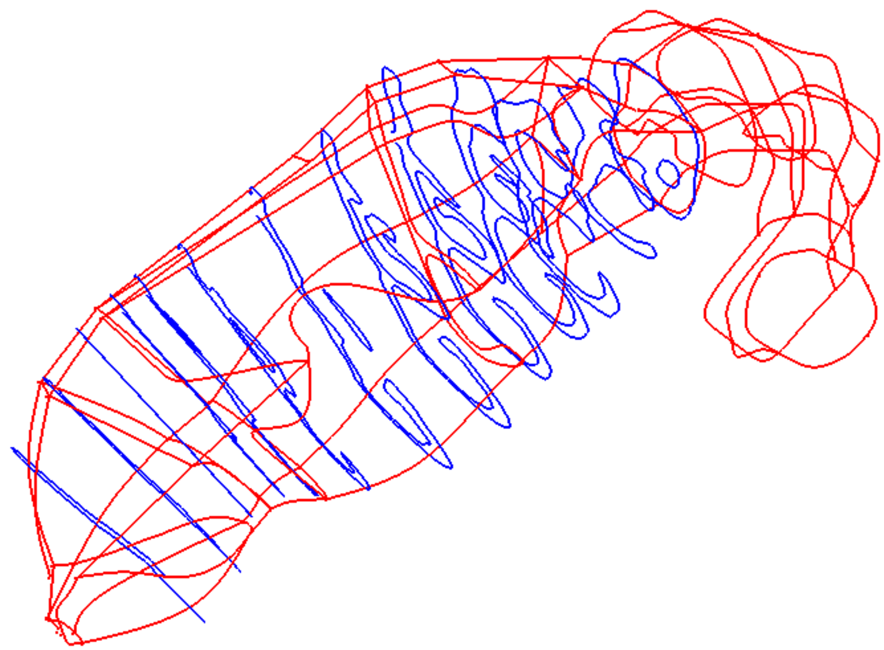


Figure 3.3 Blue curves show cross sections of subject 4 as an example. The red wireframe shows an idealized sketch drawn in OpenFOAM's BlockMesh. The idealized curve considers all subjects' common features.

Figure 3.3 shows the development of cross sections in the $+y$ direction. The entrance region is defined as the appended inlet, vestibule and valve regions. The shape of the entrance region is important, both because it presents an obstruction that yields deposition of high momentum particles and because of this region's role in guiding the flow (and lower momentum particles) toward the turbinate region.

The shape and area of the inlet of the idealized geometry were chosen to reflect those of the realistic geometries. Obviously, a different inlet could cause a different boundary condition and tend toward a wrong dynamic and hence a wrong idealized geometry. The entrance regions share similar characteristics across the realistic geometries, with some of these features having a direct effect on the particle deposition. As an example, some of the important features of entrance region can be seen in Figure 3.4.

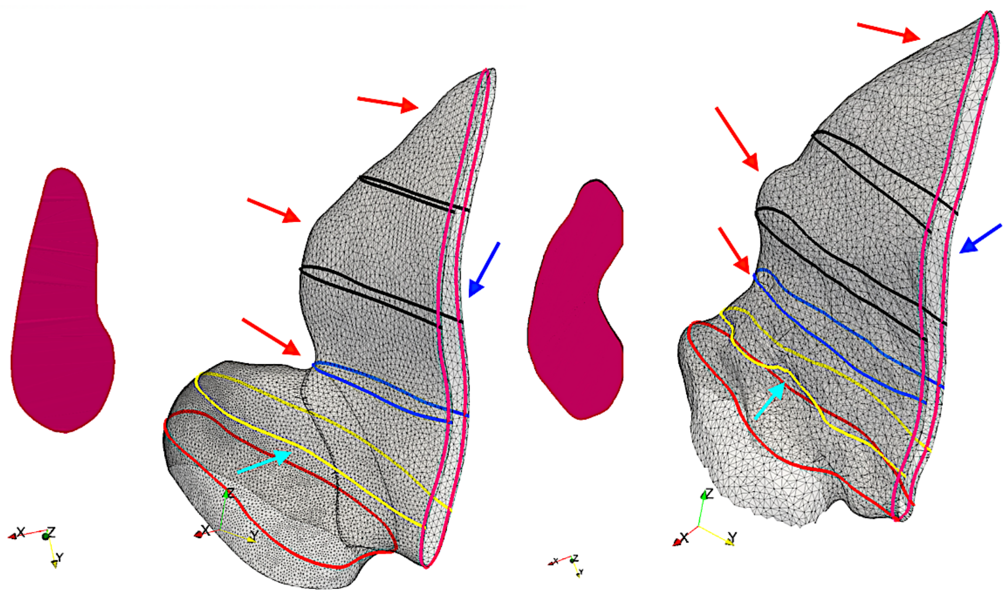


Figure 3.4 The entrance regions for subject 1 and 4 are depicted. Entrance regions in all realistic geometries show similar features. Two examples of these features are pointed out by arrows here. Note the shrinkage and expansion the red arrows illustrate in the +z direction. The blue arrow shows an important cross section between the valve and the turbinate regions. This cross section has a vertically stretched S-curve shape. The cyan arrow shows how the cross section shrinks in the vestibule-to-valve interface from the red to the yellow cross section and expands from the yellow to the blue cross section in the +z direction. The maroon-coloured section on the left of each entrance shows the inlet surface. Note the bean shape of the inlet surface.

The air flow through the nasal airway enters from the nostril and passes the vestibule region, afterward being directed toward the turbinates region through the valve. The valve is usually the narrowest part of the nasal airway geometry; hence fluid velocity increases dramatically in this region[3]. Obviously, the exit orientation of the valve would have a tremendous effect on where the flow is directed in the turbinate region.

The turbinates are known to be associated with an increase in turbulent intensity. Nevertheless, it has been pointed out by previous studies that the flow regimen of the adult nasal airway typically remains mainly laminar for common inhalation flow rates (Keyhani et al. 1995; Hahn et al. 1993; Schreck et al. 1993)

The level of geometric complexity rises dramatically at the turbinate region. For the vestibule, valve, olfactory and nasopharynx regions, the cross section mostly stays simply connected (i.e. a cross-section curve set has only one component).

The cross section shape gradually changes in the $+y$ direction. Naturally, constructing and connecting each approximated cross section and connecting them would create an interpolated idealized geometry that is also manifold[4] in two dimensions (in the local coordinate system). However, the geometry becomes more complicated in the turbinate region. The three-dimensional

---

[3] This is an obvious result for a steady incompressible flow. In this type of flow the volumetric flow rate stays nearly constant through cross sections ($\frac{dQ}{dn} \approx 0$).

[4] A surface geometry is manifold in the neighborhood of a point if in its topological space it resembles a local Euclidean space.

development of geometric structures in this region creates cross sections that are not simply connected[5].

Figure 3.5 shows the explained difference in a simplified manner. On the left case, a ribbon can be created by a set of ruled surfaces. Each ruled surface is made by connecting the corresponding points between the cross sections while procedure with similar outcome is not so trivial for the right case. An example of the aforementioned behaviour in a realistic geometry can be observed in Figure 3.6.

The appearance of the non-simply connected curves in one cross section of realistic geometry can be addressed in several ways. One solution would be to introduce an independent three-dimensional surface geometry within the turbinate region. Moreover, this surface would emulate the abrupt expansions, shrinkages and steep curves of the realistic turbinate region.

[5] In topology, simply-connected curves are often called homeomorphic. The simply-connected and none-simply-connected cross sections are topologically different (none-homeomorphic). i.e. there is no valid topological transformation between the two. Non-homeomorphism can be a source of substantial fluid mechanical (and particle deposition) differences between the surfaces constructed by these two types of cross sections.

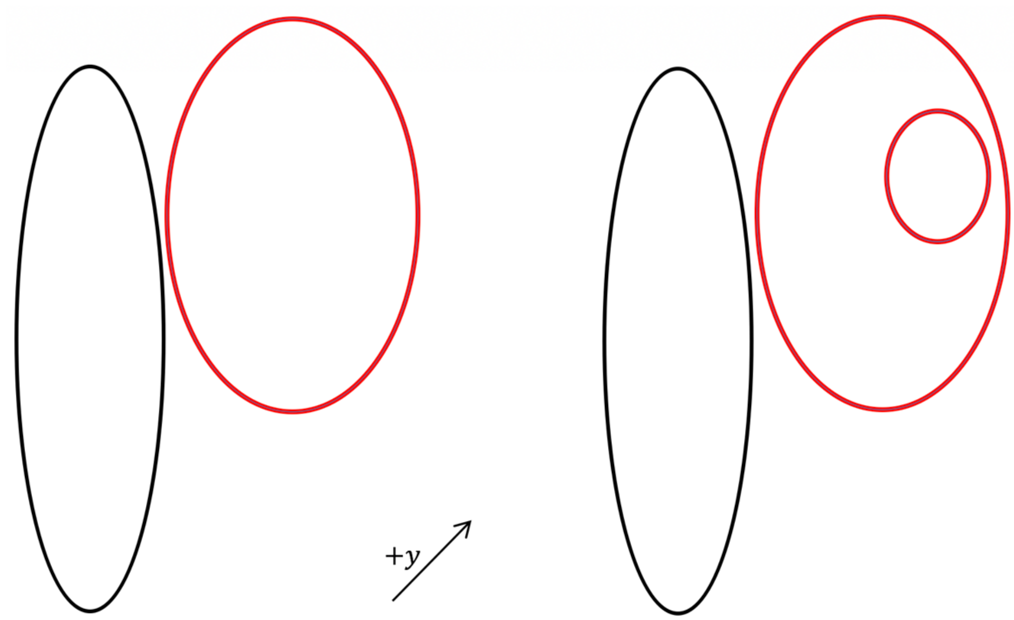


Figure 3.5 Development of black into red cross section is depicted in two simplified cases. Left shows a simply-connected cross section distorting into another simply-connected curve. Right shows a simply connected cross section followed by a non-simply-connected cross section as the geometry develops in the +y direction.

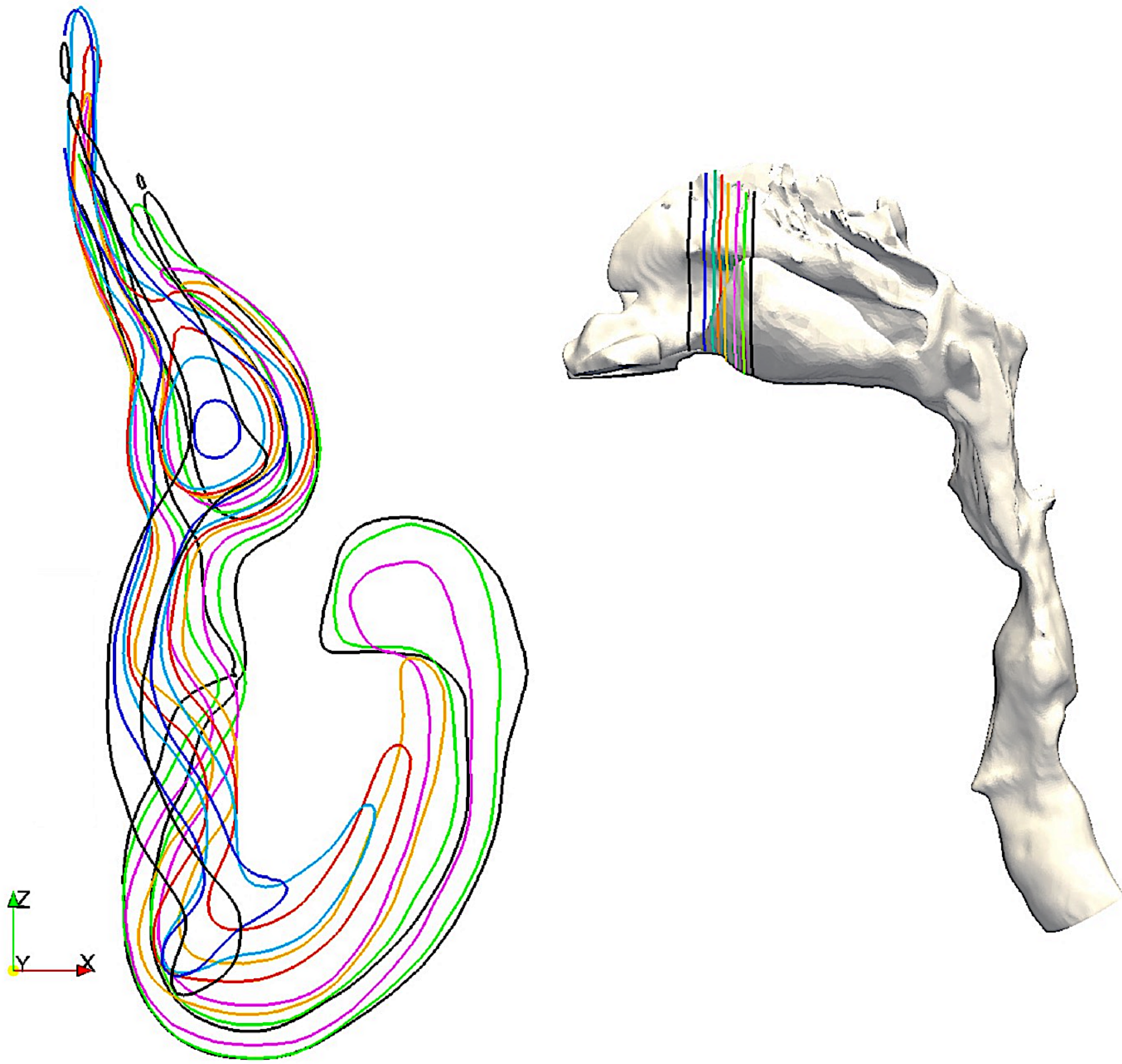


Figure 3.6 Some cross sections in the realistic geometry of subject 4 are shown. Different cross sections are shown in different colors. Note the sudden conversion of the blue cross section where it becomes non-simply connected. The unconnected portion of the curve develops further in the +y direction as the cross-section changes.

Since this object plays the role of a major obstacle in front of the flow coming from the valve, it is also referred to as such. Figure 3.7 shows the idea of the obstacle.

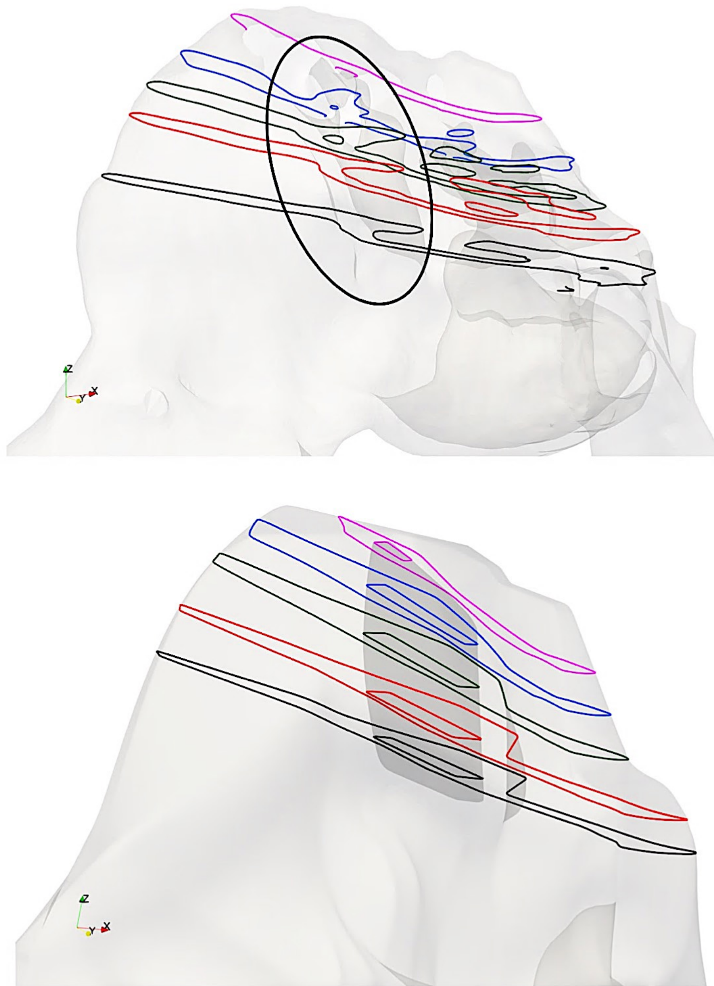


Figure 3.7 Top part of the figure visualizes subject 4 using small amount of opacity. Cross sections in different colors are from different xy planes. The obstacle structure is highlighted by the drawn black ellipse. Bottom shows an implementation of the same idea in the form of an obstacle object within the turbinate region in an idealized airway geometry.

The turbinate region in the realistic geometries shows many small-scale features in the branch. The deposition of micrometer-sized particles in the extrathoracic airway is dominated by inertial impaction. Thus, as the Stokes number increases, the probability of the deposition also increases. In simplified terms, this means smaller particles require sharper corners to deposit. The smaller scale features are expected to trap the smaller size particles. To achieve both simplicity and abruptness, the small-scale structures could be mimicked by set of generic small objects. This approach was found necessary for the idealized geometry to match average deposition in the turbinates. To this end, as a possible approach, a set of equal-sized rods has been implemented. This idea is inspired by to the widely used mesh filters to capture particles from a flow. A detailed analysis of the most efficient composition of the generic object would be rigorous and outside of scope of this study. However, in practice there is a maximum size of mesh which can be used efficiently to filter particles with certain minimum aerodynamic diameters (Kawara et al. 2016). Similarly, here the size of the rods is crucial and should be chosen small enough.

Observations from CFD results in the realistic geometries in chapter 1 also suggest an additional mechanism for particle deposition in the turbinates region. In particular, at the anterior turbinates, the flow is separated into two branches consisting of a major and minor flow. The minor flow stays nearly straight and has a smaller cross-sectional area. The major flow turns toward the side and exhibits a larger cross section. This branching of the flow partially separates the smaller particles from larger ones, with the major flow carrying only the small particles.

This particle separation mechanism resembles that in a virtual impactor. As in the case of virtual impactors, large particles follow the straight path. By contrast, small particles diverge with the major flow. In our case, a fraction of the small particles should be collected by the aforementioned rods to mimic the dynamics of realistic nasal airway turbinates. Figure 3.8 shows the basic scheme of a virtual impactor.

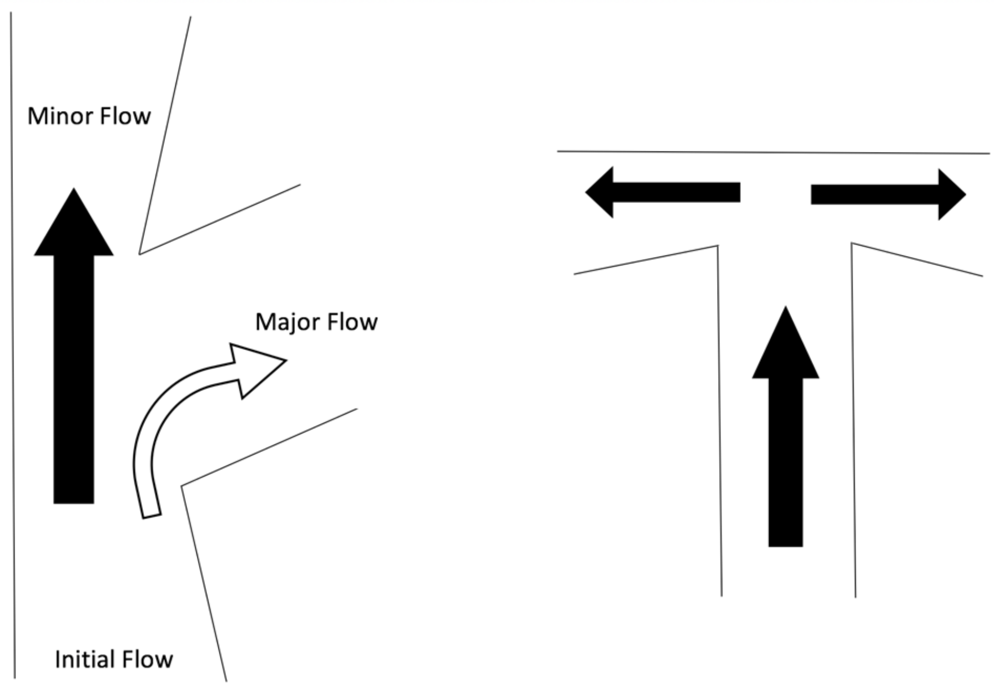


Figure 3.8 A simple sketch of a virtual impactor is shown on the left part of the figure. Note that small particles follow the major flow stream. On the other hand, a simple sketch of a conventional impactor is shown on the right. Particles may hit the obstacle based on the value of their Stokes number. Stokes number can be calculated $by\ Stk = t_r\ u_f/l_O$ in which $t_r = \rho_p\ d_p^2\ (18\mu)^{-1}$ is the relaxation time, u_f is the velocity of the fluid and l_O is the characteristic length of the obstacle.

Moreover, the boundary between the minor and major flows is an obstacle and functions as a conventional inertial impactor. The idealized geometry cross sections in the neighborhood of the turbinate obstacle and rods is depicted in Figure 3.9.

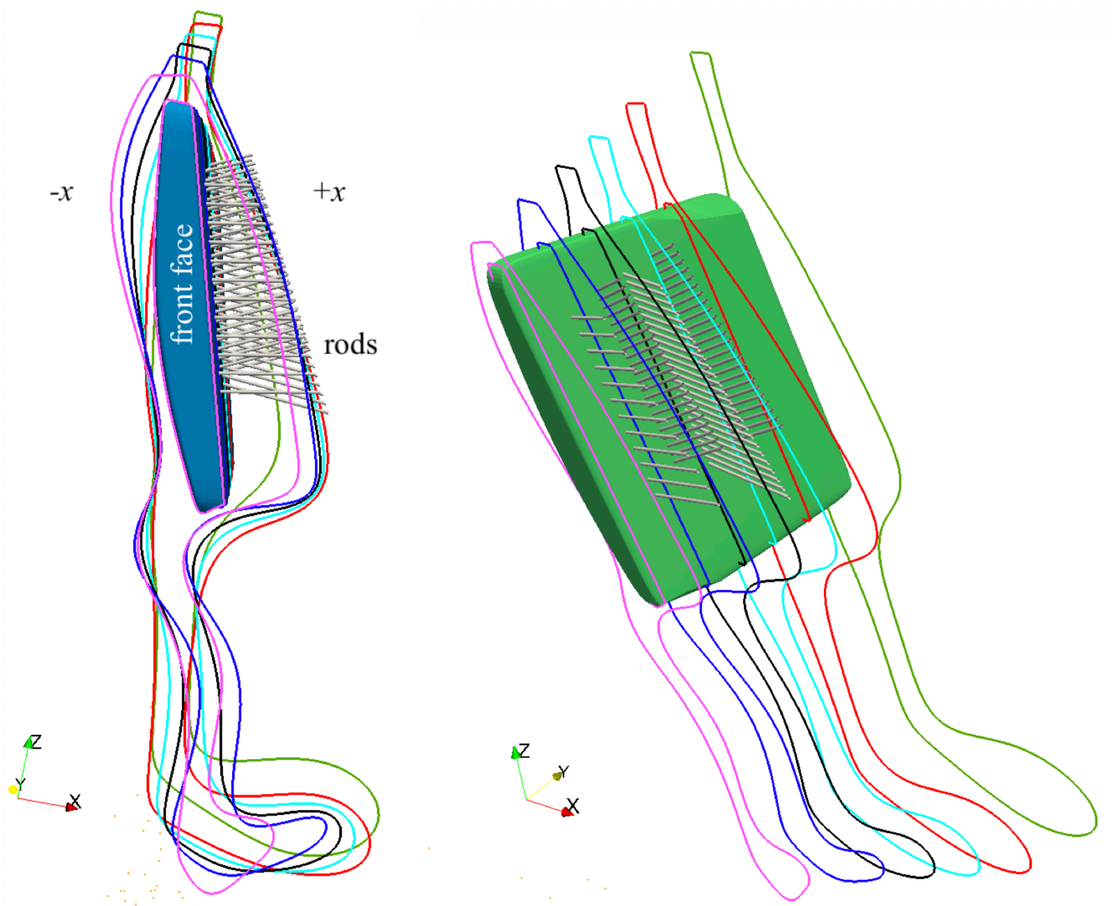


Figure 3.9 xz-plane slices of the turbinate region of the idealized geometry are shown. Different colors are assigned to different planes. The obstacle object (shown in blue on the left and green on the right) is depicted in three dimensions. The front face of the obstacle acts as a conventional impactor. Rods are shown in grey as they connect the obstacle $+x$ face to the turbinates $+x$ outer wall. These act as barriers against the small particles which are carried by the major flow.

Furthermore, the section area and overall volume were kept close to the realistic geometry and can be seen in Table 3.1.

Table 3.3 Relevant information for the 7 subjects and idealized geometry. See Figure 2.3 for approximate locations of the different listed airway regions (Vestibule, Valve, Anterior Turbinates, Posterior Turbinates, Olfactory and Nasopharynx).

Sub No	Sex	Age (years)	Airway Surface Area (cm^2)							Vol. (cm^3)
			Total Area (cm^2)	Vesti	Valve	Anter.	Poster.	Olf.	Naso.	
1	M	60	337.6	12.6	20.6	36.7	171.8	10.0	85.8	59.6
2	F	50	315.5	10.1	17.2	22.3	154.5	6.1	106.0	73.1
3	M	57	320.2	14.5	18.1	22.8	192.0	7.4	90.0	59.2
4	M	54	344.7	11.8	24.6	19.7	161.3	8.8	116.0	71.5
6	F	72	317.8	14.3	32.0	53.6	137.7	8.6	69.3	59.0
7	M	62	308.2	14.3	30.7	27.0	140.8	12.0	84.2	56.6
8	M	63	323.6	14.2	20.8	31.3	163.0	10.4	81.8	61.8
Idealized	--	--	300.7	12.6	23.8	20.8	153.0	8.6	81.9	72.4

3.2.2 Computational Fluid Dynamics of Airflow

Fluid motion in the idealized nasal airways was simulated by solving the incompressible, laminar Navier-Stokes equations. This was accomplished by using the Open-Source Field Operation and Manipulation (OpenFOAM) version 3.0.1 (OpenFOAM Foundation Ltd, UK). OpenFOAM is a collection of libraries and applications written in C++ and covers a broad range of applications in the field of scientific computing. Specifically, OpenFOAM can solve the Navier-Stokes equations of the fluid motion using the finite volume method.

OpenFOAM's BlockMesh tool was applied to automate the block generation. This was performed by creating a set of control points and edges as shown in Figure 3.10.

Each block contains eight patches. A patch is defined by four boundary curves which are created by skeleton splines. A spline is created by defining start and end points. Moreover, a spline can be adjusted by the addition of control points that create a curved edge between the start and end points.

The surface geometry of the main wall was defined as a function of chosen patches of all blocks. Analogously, the obstacle surface was constructed within the turbinate region. For simplicity, the corners of the obstacle were chosen to define a box. Additionally, splines were defined as edges of the box. Furthermore, constraints were defined to ensure the consistency of the box topology.

This measure was necessary to ensure a functional iterative process within which the shape of the obstacle was modified.

Alongside BlockMesh, most geometric manipulations were carried out using Visualization Toolkit (VTK) version 8.1.1 (Kitware Inc, USA). VTK is an open-source library for computational geometry, visualization and graphical methods. It supports various efficient and state of the art algorithms for handling several types of data structures. Moreover, VTK supports techniques for manipulation of STL files.

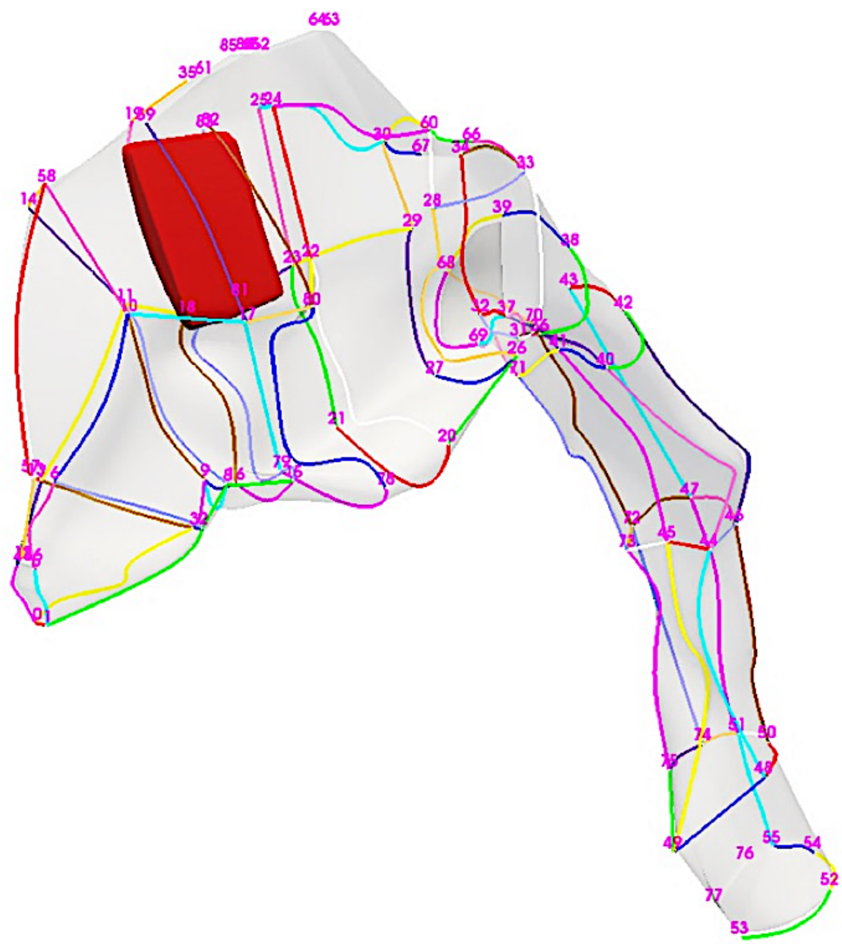


Figure 3.10 The idealized geometry is created via OpenFOAM BlockMesh tool. The red block in the middle is the obstacle and is created by using the same tool. Start and endpoints of splines are shown by numbers. Splines are curved edges connecting the points. The visualization is performed by using the ParaFOAM application.

Since both the main wall and obstacle geometries are created parametrically, they can overlap within the iterative process. Additional constraints were defined to avoid extreme cases. However, within these limits, there are numerous combinations that result in extremely sharp angles. Even a slight overlap can produce intersection edges that result in poor quality of the CFD mesh. Various in-house codes were developed in VTK to resolve this problem in an input-output (IO) approach. These codes included methods for smoothing, clipping, closing holes with distance, closing with cap, subdivision, decimation and triangulation.

VTK was also instrumental in generating the rods, which are meant to collect only the smaller particles. Hence, they are inserted from the $+x$ side of the airway and are clipped by the nearest plane of the obstacle box. This procedure was also implemented in VTK. In a trial and error approach, four rows of rods were created, each containing rods with the same y and angle γ with which they intersect the obstacle. Rods in different rows have different γ and y. Modifiable parameters for rods are the diameter of each rod d_r, the start point of the rods grid (y_s, z_s), the number of rods n_y and n_z and the distance between the center lines of the rods δ_y and δ_z in each direction.

The STL surfaces from BlockMesh underwent further repair and smoothing before being imported into OpenFOAM's SnappyHexMesh meshing tool[6]. SnappyHexMesh offers several methods to control the refinement level in specific regions. By default, refinement regions are explicitly defined for surfaces. Moreover, extra regions are added according to the calculated feature edges.

---

[6] More strictly, SnappyHexMesh does not mesh from scratch. Instead it should be provided with a mesh, e.g. from BlockMesh to produce the desired mesh by accurate boundary specifications. It accomplishes the meshing by mean of various transformations.

Feature edges belong to a class of objects in VTK that defines special edges such as boundaries and large surface normal gradients. OpenFOAM's surfaceFeatureExtract method extracts the features of the geometry.

SnappyHexMesh implements quality checks to ensure the validity of the computational grid. These checks included, but were not limited to, cell skewness, minimum volume, volume ratio, orthogonality and twist. The result was a mesh with three to five million hexahedral cells depending on the size of the feature edges set. As the size of this set increases, further refinement was essential. Figure 3.11 shows y and z clips of a sample mesh.

With the mesh prepared, the steady state flow equations were solved for velocity and pressure fields in space.

The Semi-Implicit Method for Pressure-Linked equations (SIMPLE) was used for the nonlinear outer iterations. SIMPLE is known to be an efficient solver for steady state cases. Within SIMPLE, each field is solved by a specific algorithm within the linear inner iterations. The velocity field was solved by the Gauss-Seidel method. The pressure field was solved by the Geometrically Algebraic Multigrid (GAMG) method, which uses the Diagonally Incomplete Cholesky (DIC) method. GAMG is a quick method that begins with a coarse mesh. The level of detail in the mesh increases until the convergence in the pressure field is reached.

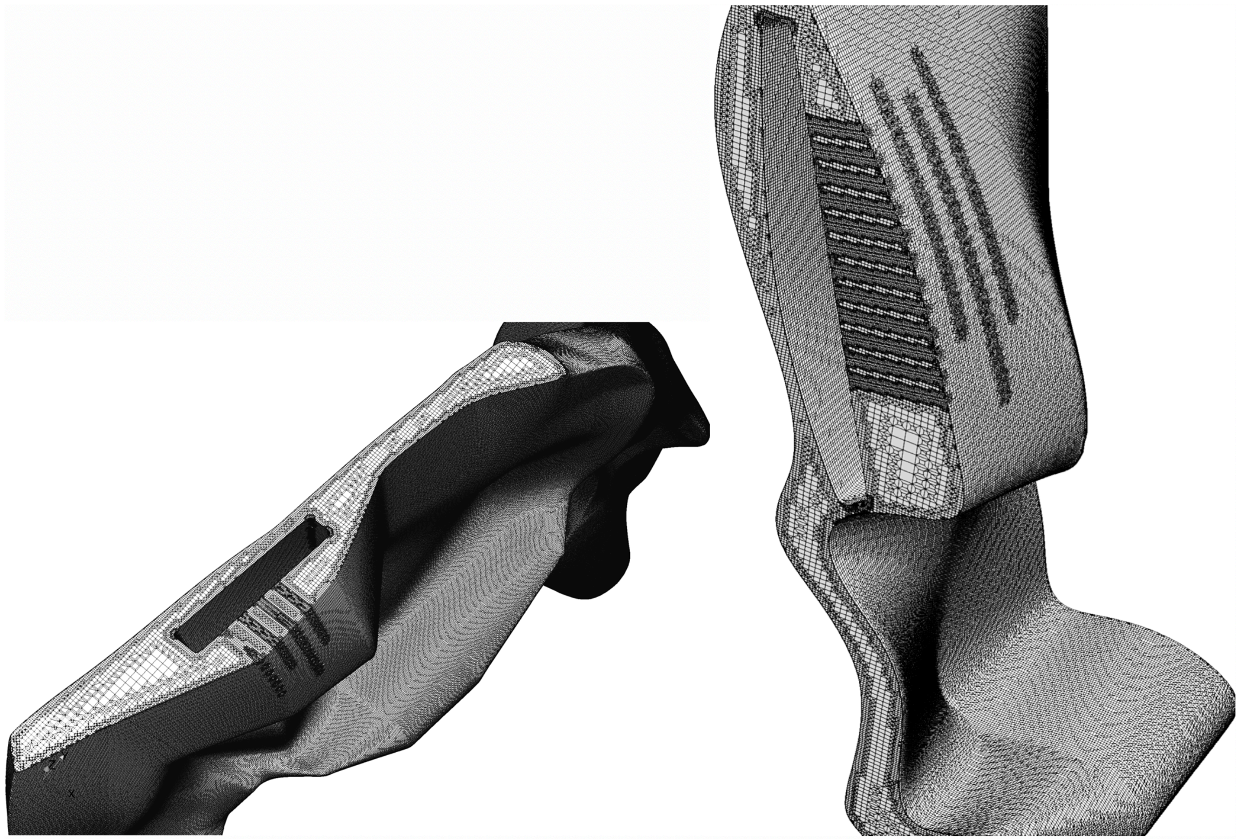


Figure 3.11 Parts of the computational grid resulting from the SnappyHexMesh tool. The hollow space created by the obstacle shows the absence of fluid in that region. The left panel shows a xy-plane clip and the right panel a xz-plane clip. The rods are seen in the mesh. Note how the mesh is refined in these regions.

Spatial discretization was second order using Gauss linear with cell limiting. Grid convergence was studied to determine the number of cells required to achieve the convergence (within 10%) for the value of the pressure drop through the airway. The boundary conditions were the same as in the realistic simulations of chapter1 and are shown in Table 3.2. Each of these conditions is defined according to boundary condition specifications provided by OpenFOAM. Special attention was paid to the boundary conditions to ensure the numerical stability of the SIMPLE iterations. The flow rate at the inlet was fixed at 15 l/min and the flow was considered to remain laminar throughout the geometry.

Table 3.2 Boundary conditions in the CFD calculations. Each italic term is a B.C. class in OpenFOAM. The *pressureInletOutletVelocity* condition is typically paired with the *totalPressure*. This is known to improve the stability of simulation by allowing the minor backflows at the outlet.

Boundary	**Pressure**	**Velocity**
Inlet	*zeroGradient* ($\frac{\partial \mathrm{p}}{\partial n} = 0$)	*flowRateInletVelocity* ($Q = 15\ L/min$)
Outlet	*totalPressure* ($p_0 = 0$)	*pressureInletOutletVelocity*
Other regions	Same as Inlet boundary	*noSlip* ($\boldsymbol{u} = \boldsymbol{0}$)

3.2.3 Lagrangian Particle Tracking

After solution of the velocity and pressure fields, particle tracking was performed. The particles were assumed to be non-evaporating and were assumed to stick to all boundary surfaces. Stuck and escaped particles were labeled by OpenFOAM and were no longer updated during the rest of the particle tracking iterations. This approach saves considerable amount of memory and computational power.

The momentum of the particles was assumed to be one-way coupled with that of the fluid. In other words, the particles do not disturb the flow. Particle position was updated using Newton's second law

$$m_p(\frac{\partial u_p}{\partial t})^i = S_p(u_f - u_p^i) + S_u \tag{3.1}$$

where S_u and S_p are called the overall explicit and implicit contributions to the particle force at time-step i^{th} on the p^{th} particle respectively. In steady one-way coupling, the velocity field u_f remains constant at a given location, with velocity-dependent forces acting on the particle as an implicit drag force. The gravity-dependent buoyancy force is an explicit contribution[7].

The velocity of the fluid was interpolated to the location of the particle using the linear cell method. The implicit Euler method was used for time integration of particle trajectory. The implicit Euler method is known to be unconditionally stable; however, due to nonlinearity of the flow field, extra caution is exercised by performing time step size analysis. Furthermore, a convergence analysis of the number of injected particles was conducted to ensure that the number of particles was satisfactory. The drag coefficient employed was the Schiller-Neumann (equation 2.1), and the viscosity of the air was set to $\nu = 1.5 \times 10^{-5} m^2/s$.

[7] Terms and notations in this equation are strictly following the ones used in OpenFOAM's source code and documentation.

Lagrangian particle tracking was accomplished using the IcoUncoupledKinematicParcelFoam (IUKPF) application of OpenFOAM. IUKPF was further customized by compiling a local code via OpenFOAM's WMake utility. IUKPF utilizes a simplified version of the general Kinematic Cloud (KC) objects for particles and assumes them to be uncoupled with respect to each other. The main OpenFOAM required dictionary file name is KC-Properties and contains necessary values used by IUKPF. Since IUKPF is a very simplified particle tracking method, most entries of KC-Properties were not set.

To explore the possibility of different injection positions and their impact on the deposition results, the tip of the particle spray injection was placed at various locations. One approach in this regard was to specify random locations within the entrance region and average the deposition results among all these random locations. In order to generate random locations for the tip of the injector, a VTK location generator code was used. Particles were injected within the nostril from a planar disk region with 1mm diameter; the position of the disk was varied within the nares to define 200 random positions.

Particle injection occurred at a constant velocity. Because of one-way coupled assumption, the injection volumetric flow rate did not determine anything and was arbitrary. In other words, the values of initial velocity and number of the particles were utilized only to identify the initial condition of the particles. The injection location was varied from a little inside the entrance of the nares to a little after the entrance of the nasal valve region, with these insertion depths varying approximately in the range of 0.1 to 1.5 cm from the inlet. Figure 3.12 shows the randomly generated injection disks within the nares.

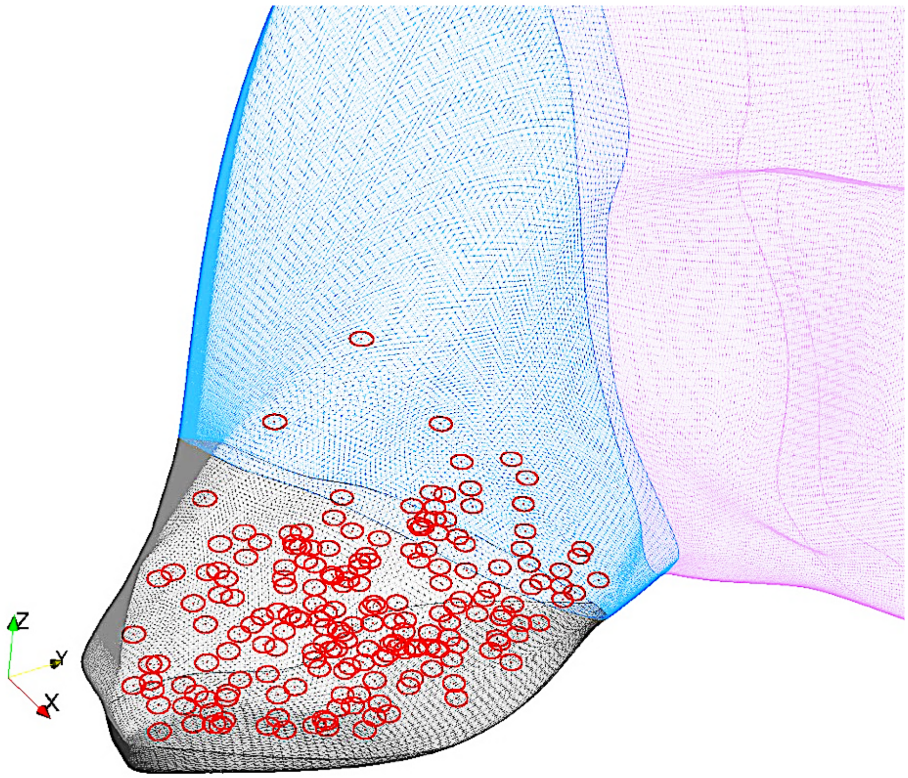


Figure 3.12 Positions of particle injection at the entrance are shown. Circles show the location and alignment of the tip of the injector. Centers of circles were randomly chosen and were offset a minimum of 1mm from the walls. Particles were introduced randomly on the surface of each disk. The injection half-cone inner and outer angles were set at 0° (+z direction) and 15°. The injection direction for an individual particle is interpolated between the inner and outer half cone angle based on the location at which it appears on the injection disk.

Ten thousand particles were injected through each disk, with the particles initial velocity in the $+z$ direction. Particle injection velocities produced a cone shape with specified inner and outer angles of 0 and 15 degrees. These parameters were chosen from a subset of the ones used for the realistic geometry simulations in chapter 1 and they were based on common practice with inhaler devices.

Table 3.3 shows the combination of parameters used for particle tracking simulations in this study. Because of the iterative process used in designing the idealized geometry, the number of injection positions was kept low during iteration of the geometry shape; a total of 80 particle tracking simulations per idealized geometry parametrization were performed. However, for the final idealized geometry, 4000 simulations were performed. The latter is consistent with the parameter set used in the realistic geometry study of chapter 1.

Table 3.3 Particle parameters. These are used in idealized geometry particle tracking simulations. For the validation case the number of particle tracking cases is 4000.

Parameter	**Number of Parameter**	**Values**
Particle diameter	5	$\{5, 10, 15, 20, 40\}$ microns
Injection cone angle	2	0° inner and 15° outer
Injection direction	1	Upward $(+z)$
Particle injection velocity	4	0-20 m/s
Position of injection disk (Only for the validation case)	200	Random on $1mm$ diameter disk

Several Bourne Again Shell (Bash) scripts were developed to detect idle CPU threads for use in simultaneous particle tracking simulations. The simulations were done in parallel on a local Beowulf cluster[8] which is based on the Network File System (NFS) (Sun Microsystem, USA) protocol. This cluster contains 20 Threads overclocked at 4.2 GHz, 24 threads 2.4 GHz and 4 threads at 3.5 GHz, for a total of 48 threads. Memory in use was 200 GB. To calculate the regional deposition, the geometry was divided into the regions of vestibule, valve, turbinates, olfactory, nasopharynx and outlet. Additionally, the turbinates were subdivided into main wall, rods and obstacle. The diagram of the full iterative process is shown in Figure 3.13

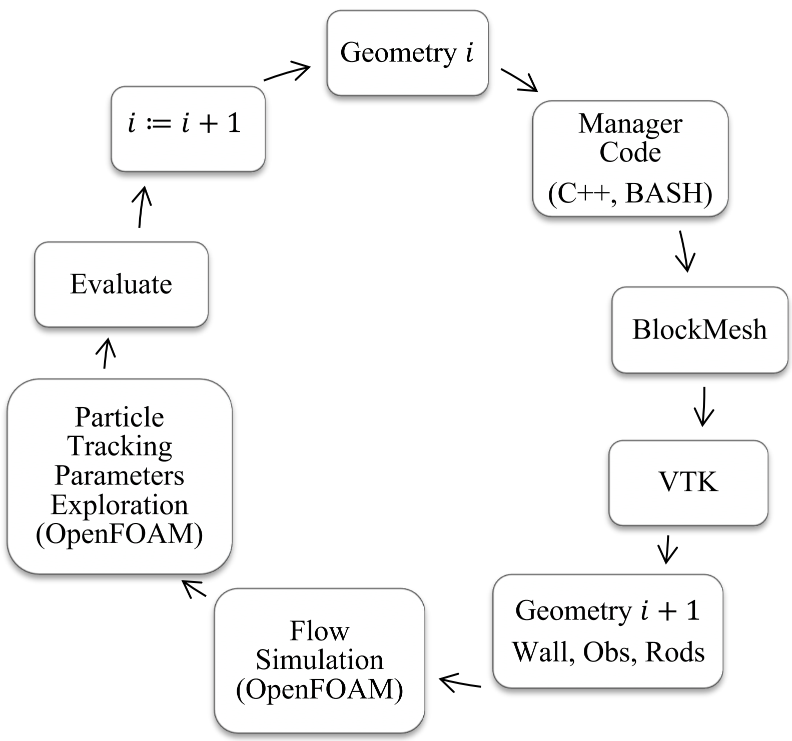


Figure 3.13 The complete iterative procedure used in the development of the idealized airway.

[8] A Beowulf cluster is a cluster of consumer-grade computers that uses Local Area Network (LAN) protocols to share processing, memory and storage among them.

3.2.4 Evaluation of the Quality of an Idealized Geometry

In this study the development of the idealized geometry was based on a few iterations that involved starting with an initial idealized geometry and iteratively distorting this geometry with the aim of achieving a closer and closer match to average deposition seen in the different regions of the realistic geometries. In order to evaluate each geometry modification, a norm is needed. For this purpose, let $\mathbf{C}$ be a functional associated with the CFD results and $\mathbf{L}$ a functional associated with the Lagrangian particle tracking fields, $\mathbf{M}$ is the functional representing a regional average deposition matrix, with its rows based on particle diameters and its columns based on particle initial velocity:

$$\mathbf{M}(\mathbf{x}) = \mathbf{L}\Big(\mathbf{C}\big(\mathbf{S}(\mathbf{x})\big)\Big)$$

With the formalism, an objective function can be defined as $\mathbf{f}(\mathbf{x}) = \left|\mathbf{M}(\mathbf{x}) - \mathbf{M}_{ref}\right|$ in which $|\,.\,|$ denotes a norm, and the optimization problem in design space $\boldsymbol{\Gamma}$ devolves to finding $\mathbf{S}$:

$$\arg\min_{\mathbf{x}\in\boldsymbol{\Gamma}} \mathbf{f}(\mathbf{S})$$

The latter equation describes a multi-objective optimization problem; i.e. there is no unified solution capable of minimizing all components of $\mathbf{f}$ simultaneously. Using the weighted scalarizing method and Einstein notation, the objective function can be represented as

$$\mathbf{f}(\boldsymbol{x}) = w^j \sqrt{\sum_{u=1}^{4}\sum_{d=1}^{5}\left(m_{du}^{j}(\mathbf{x}) - m_{du_{ref}}^{j}\right)^2}$$

By assuming $w^j = 1$, the previous expression becomes the sum of L^2 norms of the regional deposition. This value of the norm provides a measure for evaluating whether or not a given realization of the idealized geometry is close to giving the target values of average deposition in the realistic geometries.

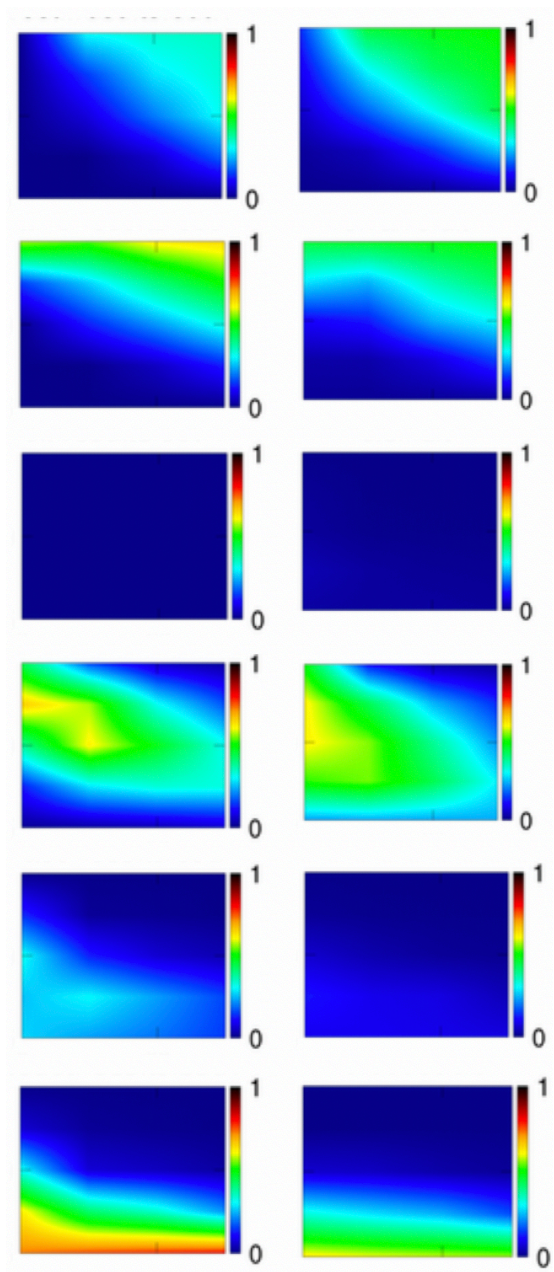
3.3 Results and Discussion

3.3.1 Monolithic Surface

The main wall of the surface geometry was constructed by arranging skeleton splines. The number of possible geometries is infinite. The cross sections are homeomorphic in this case. A slight modification in a control point of a spline results in a smooth change of the surface geometry. Through iterations over the control points of the main wall, many geometry versions are created. Figure 3.14 shows one of these geometries that provides reasonable regional deposition values. Figure 3.15 shows the regional deposition results. Although the overall behaviour is good, the small particles are not captured in the turbinates. Consequently, the nasopharynx and outlet experience more particle deposition and escape, respectively. This behaviour contradicts the average behaviour in the realistic geometries which is shown in the right column of Figure 3.15.

Figure 3.14 The Y-shaped cross sections of the idealized geometry are shown on the left. The surface of the geometry is shown on the right. This specific geometry is called the monolithic surface because (1) the geometry is made solely with sequences of blocks in BlockMesh and (2) the cross sections remain homeomorphic with respect to each other. The cross sections in colors show the simply-connected behavior of curves in the turbinate region of this geometry.

Figure 3.15. Each row denotes a certain region (in order: Vestibule, Valve, Olfactory, Turbinates, Nasopharynx, Outlet). The deposition fraction in the monolithic idealized geometry (plots in left column) and averaged over the realistic geometries (plots in right column) from chapter 1 are shown. The vertical axis in each plot denotes the particle diameter (5-40 micron) while the horizontal axes are the particle initial velocities (0-20 m/s). Note that small particles are not well captured at lower spray velocities by the turbinate region of the idealized geometry in this case. The color scale is interpolated and shows the deposition fraction (0-1) out of total particles.



3.3.2 Rods

With the intention of capturing more of the small particles, rods were introduced in the turbinates of the geometry. After careful evaluation of different sizes of rods, a diameter of 0.2 mm was chosen. Figure 3.16 shows a grid of rods distributed over the idealized geometry aligned on the x axis. As shown in Figure 3.17, turbinate deposition improved for the small particles. However, too many mid-sized particles were deposited in the turbinates. This suggests that adding a mechanism capable of separating the particles by size could be used to improve the regional deposition results.

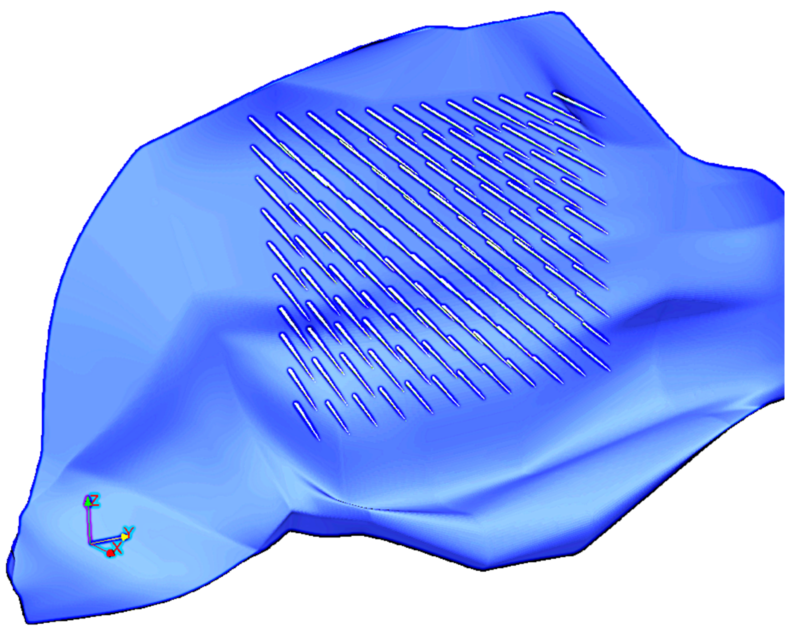
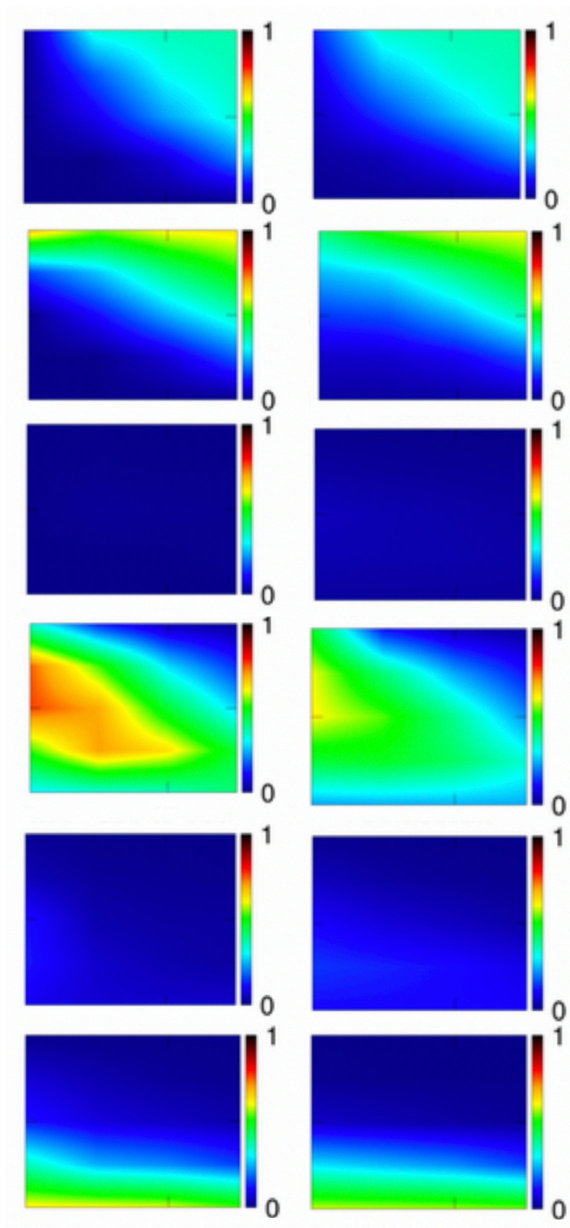


Figure 3.16 A penultimate version of the idealized geometry is shown. A grid of rods (shown in the brighter color) is penetrates the turbinate region side. The rods protrude in the x-direction across the full breadth of the turbinates airway.

Figure 3.17. Each row denotes a certain region (in order: Vestibule, Valve, Olfactory, Turbinates, Nasopharynx, Outlet). The deposition fraction in the idealized geometry with rods (plots in left column) and averaged over realistic geometries (plots in right column) are shown. The vertical axis in each plot denotes the particle diameter (5-40 micron) while the horizontal axes are the particle initial velocities (0-20 m/s). Note that particle deposition is too great in the turbinates in this case. The color scale is interpolated and shows the deposition fraction (0-1) out of total particles.



3.3.3 Virtual Impactor

Two impactor type deposition mechanisms were inspired by observations of the realistic geometries. In particular, the previously noted obstacle feature (Figure 3.9) had a trivial equivalence in the complex realistic geometries, and small scale geometric traps for the smaller particles were mimicked using small rods in the idealized geometry. Adding the obstacle in the middle of the turbinate region created two paths, resulting in a virtual impactor, while adding the rods to the major flow branch further improved the regional deposition. Figure 3.18 shows the regional deposition results for this case. Since the results were a very good match, the same 200 random injection positions at the entrance in the realistic geometries were then applied to this final geometry. As a result, the deposition matrices[9] smoothed further and resulted in nearly identical turbinate deposition. Figure 3.19 shows the deposition values in the final idealized geometry versus all realistic geometries. The deposition in the idealized geometry is typically in the middle of the range of those in the realistic geometries. While certain subjects do have deposition that is reasonably close to the average of all subjects in various regions, no single subject matches average deposition accurately in all regions for all parameter values.

For further validation, many of the cases were visualized through animations, which also verified the explained behaviour of conventional and virtual impactor mechanisms. The animations were made utilizing OpenFOAM's ParaFOAM application, an extension of Paraview (Kitware Inc, USA) visualization software. Paraview is based on VTK.

---

[9] Each average deposition matrix corresponds to a region in the geometry. Rows are particle sizes and columns are particle initial velocities. There are six deposition matrices for each geometry. Each matrix has five columns and four rows (twenty components).

Figure 3.18. Each row denotes a certain region (in order: Vestibule, Valve, Olfactory, Turbinates, Nasopharynx, Outlet). The deposition fraction in the virtual impactor idealized geometry (plots in left column) and averaged over realistic geometries (plots in right column) are shown. The vertical axis in each plot denotes the particle diameter (5-40 micron) while the horizontal axes are the particle initial velocities (0-20 m/s). The color scale is interpolated and shows the deposition fraction (0-1) out of total particles.

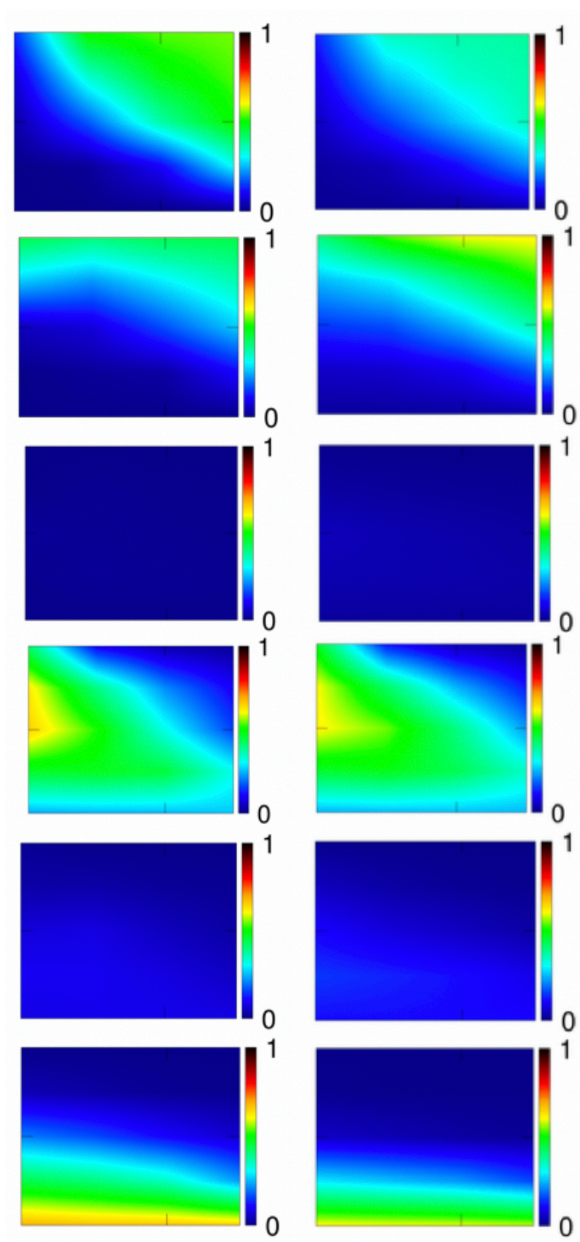
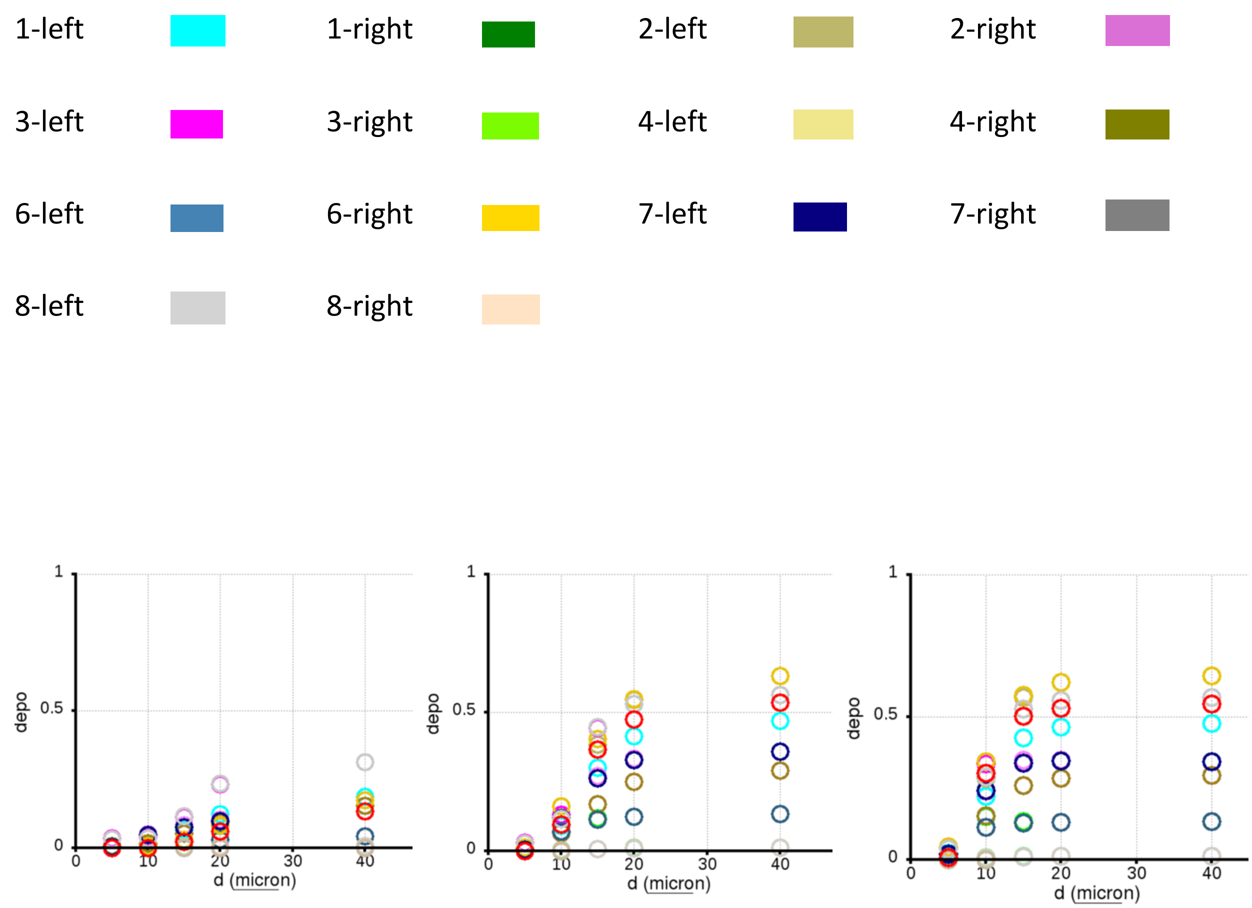
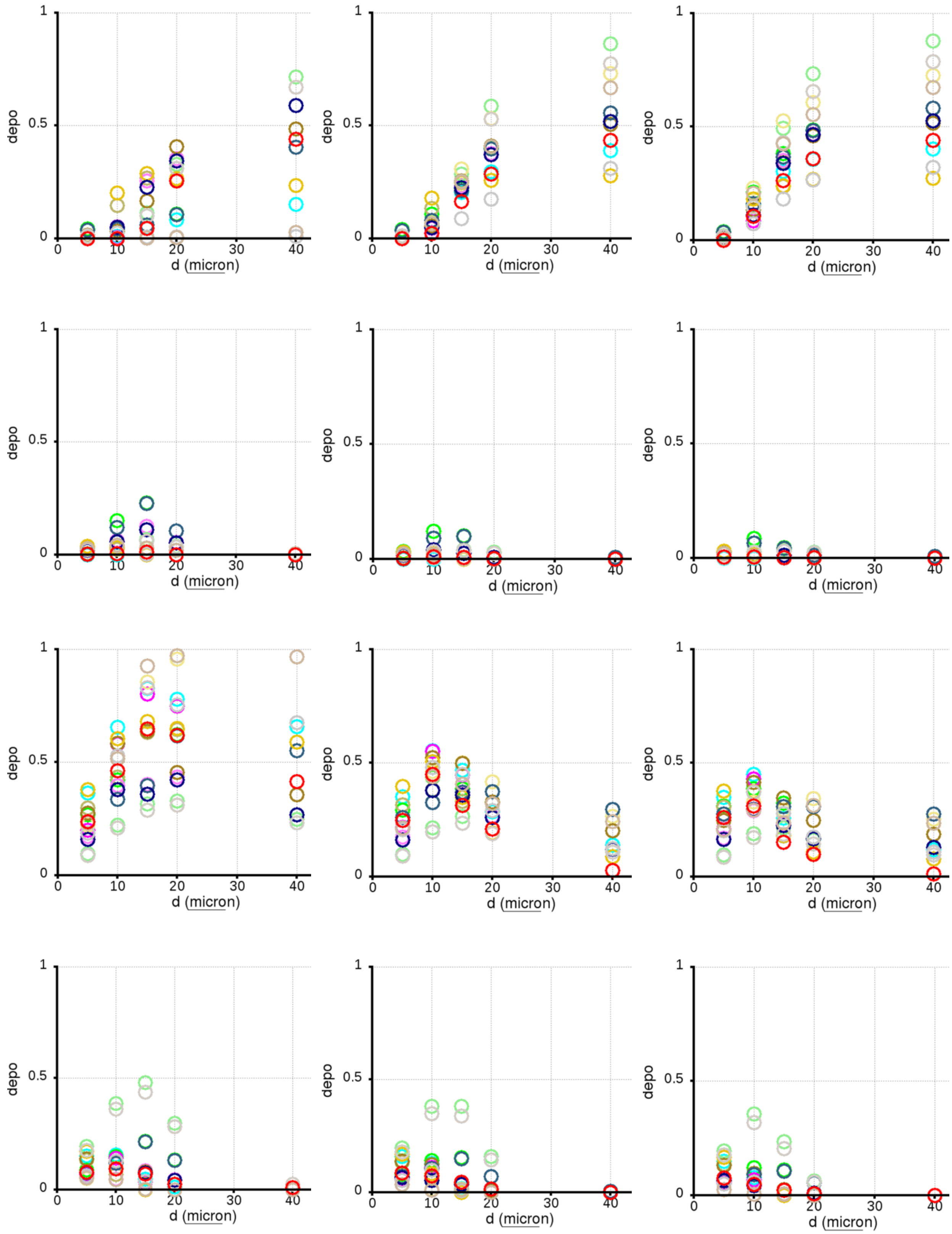
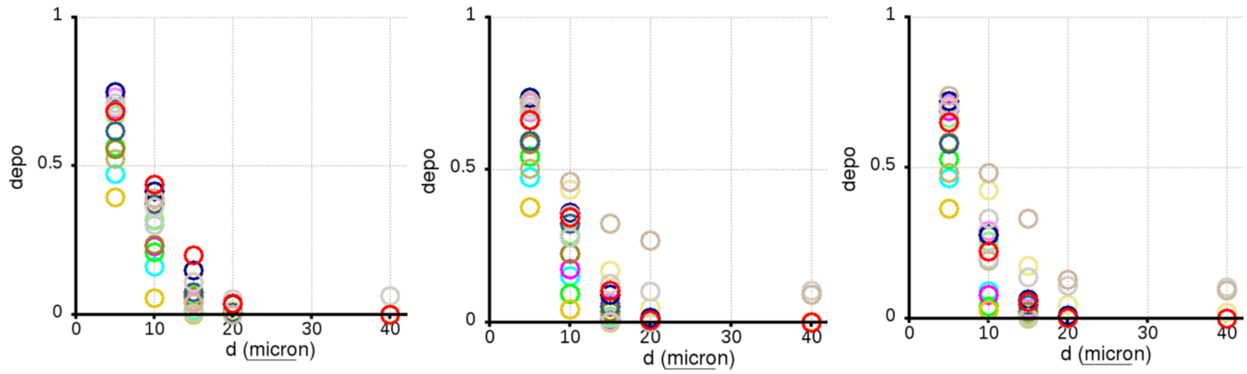


Figure 3.19. Each triple plot in a row denotes a certain region (in order: Vestibule, Valve, Olfactory, Turbinates, Nasopharynx, Outlet). Each column shows an initial particle velocity (from left to right 0, 20 and 40 m/s). The color markers show average regional deposition in different individual realistic subjects (from chapter 1) while the red marker shows the regional deposition in the final idealized geometry. The vertical axis is the fraction (0-1) of 10000 particles. The data is averaged over 200 injection locations defined randomly within entrance region.




depo
d (micron)
0
0.5
1
0
10
20
30
40



3.3.4 *Further Discussion*

The largest deposition in the turbinate regions occurs for to intermediate particle sizes. This result is in agreement with CFD simulations in chapter 1 as well as with the majority of cases studied by others (Keeler et al. 2015).

(Keeler et al. 2015). Turbinate deposition is also largest for zero spray velocity. This result is explained by the fact that if the particle is too large, or its velocity is too high, it will impact the entrance wall due to high inertia. In the opposite case, particles will penetrate and escape the outlet. Hence the zero-velocity intermediate sized particles are the ones deposited in the turbinate region. The average olfactory deposition was nearly zero, as expected. This result was previously reported by previous studies (Kiaee et al. 2018; Xi et al. 2016; Schroeter et al. 2006). Penetration remained mostly as observed in the average realistic geometries.

Two main impactor mechanisms were necessary to mimic deposition in the turbinate region. Conventional impaction is the main mechanism responsible for the medium and large particle deposition occurring at front face and $-x$ side of the obstacle. However, a virtual impactor mechanism functions at $+x$ the side of the obstacle. A fraction of the remaining small particles that escaped the obstacle were deposited on the rods on this side. The 0.2 mm diameter chosen for the rods was near the optimum value for collecting the small particles. Larger diameters (e.g. 1 mm) tended to disturb the flow too much causing particles to follow a path around the larger rods. On the other hand, smaller rods diameters would make manufacturing more difficult. Furthermore, the rods' angle of inclination plays an effective role in collecting more of small particles. The angles achieve this goal by reducing the rods' overlap.

3.3.5 Optimization Framework

The overall structure used in this study was part of a numerical optimization framework. The complete framework was built upon parametric geometries and was successfully tested. The iterations themselves were performed within this optimization framework. However, because of numerous local minimums and an extremely expensive objective function that required meshing, flow simulation and several particle tracking for each evaluation, it was not possible to perform a successful optimization convergence during the current study. Nevertheless, with the optimization framework in place, it may be possible to achieve the full optimization loop if enough interactive[10] computational resources were made available.

This optimization framework was established using Dakota (Sandia Labs, USA) software. Dakota is an optimization solver under active development by Sandia National Labs since 1997 and is

---

[10] The computational resource is needed to remain interactive. This is due to the requirement of manual verifications with regard to the topologic validity. Furthermore, many VTK applications were modified as soon as a local minimum was passed.

written in Fortran and C++. It provides an interface between solvers and external iterative methods. Dakota works by setting the input file "*dakota.in*".

A vector of design parameters **x** was defined. Using the definition of objective function as before, the constraints were

$$\mathbf{Gx} = \mathbf{0}$$

$$\mathbf{a} < \mathbf{Ax} < \mathbf{b}$$

in which **G** and **A** are the equality and inequality constraint matrices. Furthermore, and **a** and **b** are lower and upper bound vectors respectively. By using Taylor expansion, a second order Newton method can be written as

$$\mathbf{f}_{n+1} \approx \mathbf{f}_n + \nabla \mathbf{f}_n^T \Delta\mathbf{x}_n + \frac{1}{2}\Delta\mathbf{x}_n^{\mathrm{T}}\, \mathbf{Hf}\, \Delta\mathbf{x}_n$$

in which **H** is the Hessian of operator on **f**. This method requires inversion of the Hessian matrix at each iteration. Evaluation of **f** requires computational resources on the order of teraflops, resulting in extreme computational cost. Hence, a Quasi-Newtonian method (Hessian-free method) is beneficial. A general Quasi-Newtonian method follows an approximation of Hessian method instead. If we denote this estimation as **B**, the following constraint needs to be satisfied

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{B}_n^{-1}\nabla\mathbf{f}_n$$

Furthermore, the implicit calculation of gradients is also expensive and will therefore not be requested externally by the optimization procedure. The optimization tool uses a forward difference method to calculate the gradient vector explicitly. The merit function is the Argaez-

Tapia function. A line search method in this way is called value based and only satisfies the sufficient decrease condition.

If **f** is convex or nearly convex, a reasonable number of iterations would result in convergence. As noted, the above optimization method was implemented, but did not successfully converge.

3.4 Conclusions

The aim of this study was to use computational methods to develop an idealized nasal airway geometry capable of mimicking the regional deposition pattern observed in a set of realistic geometries. Regional deposition in the idealized geometry was found to be in good agreement with the median of that seen for regional depositions in the realistic geometries. The present idealized geometry may be as a useful benchtop tool for *in vitro* research and development of nasal spray formulations.

Chapter 4: Conclusions

4.1 Summary

This thesis was divided in two parts, with the overarching objective to provide an idealized adult nasal airway geometry. This geometry was intended to mimic the average regional nasal deposition pattern in adults using pharmaceutical nasal sprays.

During the first part of the thesis, described in Chapter 2, a comprehensive computational parameter exploration was performed. Using computer simulations, regional particle deposition was calculated over a wide range of parameters. These results were substantiated through comparison with previous experimental and computational studies. A particular focus of Chapter 2 was to explore combinations of parameters that targeted deposition to the olfactory region. Although with a specific combination of parameters (which included a very localized droplet injection location) deposition as high as 100 percent was observed in the olfactory region of some subjects, the average deposition was very low. Furthermore, olfactory deposition was found to be highly variable between different realistic nasal airway geometries. When averaged over all injection locations, maximum olfactory deposition ranged over two orders of magnitude between geometries. This level of intersubject variability in dosing poses a significant obstacle to the development of nose-to-brain drug delivery devices that target olfactory deposition.

The second part of thesis, described in Chapter 3, was focused on the design and development of an idealized adult nasal airway geometry. In numerical simulations, this idealized geometry was able to mimic the average deposition observed in realistic geometries reported in Chapter 2. The idealized geometry has the potential to be used as a reference geometry in modelling, simulations and experiments performed using pharmaceutical nasal sprays and other intranasal drug delivery devices.

4.2 Future Work

Although the thesis work presented herein met the goal of developing an idealized adult nasal airway geometry for testing pharmaceutical nasal sprays, several aspects might be refined or explored further in future work. First and foremost, confirmation through *in vitro* experiments that deposition in the proposed idealized geometry predicts average deposition in the realistic geometries is warranted. This confirmation will be an important and necessary step before the idealized geometry proposed here is adopted for wider testing.

Additionally, although the methodology adopted in Chapter 3 did ultimately result in a satisfactory idealized geometry (as assessed by numerical simulation), depending solely on qualitative feature extraction and manual simplification could have resulted in a sub-optimal geometry, especially when a larger number of subjects is involved. More rigorous methods are available to provide finer settings within the design space. Gradient-based optimization and artificial neural networks are both widely utilized for shape optimization in many fields (Bandara et al. 2016; Masters et al. 2016; Kim 2006; Song and Keane 2004; Yildiz et al. 2003; Song et al. 2002). A quasi-Newtonian method demanding low recourses has also proven robust and affordable (Andrew 2008; Xu and Zhang 2001). In the present thesis a complete functional optimization framework was built. Although it did not produce a converged optimization loop, it prepares the ground for future attempts. The evaluation function in this study was extremely expensive. Furthermore, because several nonlinear geometric features affected the particle dynamics in realistic airways, several local minimums were possible. This means the complete iterative process would require extreme computational power. Nevertheless, there are qualitative observations which could improve the overly simplified surface, bypass many local traps and potentially converge to the desired optimal case.

Bibliography

Al-Ghananeem, A.M., Sandefer, E.P., Doll, W.J., Page, R.C., Chang, Y., and Digenis, G.A. (2008). Gamma scintigraphy for testing bioequivalence: A case study on two cromolyn sodium nasal spray preparations. *Int. J. Pharm.,* 357(1–2):70–76.

Andrew, G. (2008). Overview of Quasi-Newton optimization methods. *Numer. Optim.,* 1–5.

Bahadur, S. and Pathak, K. (2012). Physicochemical and physiological considerations for efficient nose-to-brain targeting. *Expert Opin Drug Deliv,* 9(1):19–31.

Bandara, K., Rüberg, T., and Cirak, F. (2016). Shape optimisation with multiresolution subdivision surfaces and immersed finite elements. *Comput. Methods Appl. Mech. Eng.,* 300:510–539.

Bates, A.J., Doorly, D.J., Cetto, R., Calmet, H., Gambaruto, A.M., Tolley, N.S., Houzeaux, G., and Schroter, R.C. (2015). Dynamics of airflow in a short inhalation. *J. R. Soc. Interface,* 12(102):20140880.

Below, A., Bickmann, D., and Breitkreutz, J. (2013). Assessing the performance of two dry powder inhalers in preschool children using an idealized pediatric upper airway model. *Int. J. Pharm.,* 444(1–2):169–174.

Bousquet, J., Khaltaev, N., Cruz, A.A., Denburg, J., Fokkens, W.J., Togias, A., Zuberbier, T., Baena-Cagnani, C.E., Canonica, G.W., Van Weel, C., Agache, I., Aït-Khaled, N., Bachert, C., Blaiss, M.S., Bonini, S., Boulet, L.P., Bousquet, P.J., Camargos, P., Carlsen, K.H., Chen, Y., Custovic, A., Dahl, R., Demoly, P., Douagui, H., Durham, S.R., Van Wijk, R.G., Kalayci, O., Kaliner, M.A., Kim, Y.Y., Kowalski, M.L., Kuna, P., Le, L.T.T., Lemiere, C., Li, J., Lockey, R.F., Mavale-Manuel, S., Meltzer, E.O., Mohammad, Y., Mullol, J., Naclerio, R., O'Hehir, R.E., Ohta, K., Ouedraogo, S., Palkonen, S., Papadopoulos, N., Passalacqua, G., Pawankar, R., Popov, T.A., Rabe, K.F., Rosado-Pinto, J., Scadding, G.K., Simons, F.E.R., Toskala, E., Valovirta, E., Van Cauwenberge, P., Wang, D.Y., Wickman, M., Yawn, B.P., Yorgancioglu, A., Yusuf, O.M., Zar, H., Annesi-Maesano, I., Bateman, E.D., Kheder, A. Ben, Boakye, D.A., Bouchard, J., Burney, P., Busse, W.W., Chan-Yeung, M., Chavannes, N.H., Chuchalin, A., Dolen, W.K., Emuzyte, R., Grouse, L., Humbert, M., Jackson, C., Johnston, S.L., Keith, P.K., Kemp, J.P., Klossek, J.M., Larenas-Linnemann, D., Lipworth, B., Malo, J.L., Marshall, G.D., Naspitz, C., Nekam, K., Niggemann, B., Nizankowska-Mogilnicka, E., Okamoto, Y., Orru, M.P., Potter, P., Price, D., Stoloff, S.W., Vandenplas, O., Viegi, G., and Williams, D. (2008). Allergic Rhinitis and its Impact on Asthma (ARIA) 2008 update (in collaboration with the World Health Organization, GA2LEN and AllerGen). *Allergy Eur. J. Allergy Clin. Immunol.,*.

Byron, P.R., Hindle, M., Lange, C.F., Longest, P.W., McRobbie, D., Oldham, M.J., Olsson, B., Thiel, C.G., Wachtel, H., and Finlay, W.H. (2010). *In Vivo–In Vitro* Correlations: Predicting Pulmonary Drug Deposition from Pharmaceutical Aerosols. *J. Aerosol Med. Pulm. Drug Deliv.,* 23(S2):S-59-S-69.

Canada, H. (2006). Pharmaceutical Quality of Inhalation and Nasal Products. *Heal. Canada,* 2(613).

Chen, J.Z., Katz, I.M., Pichelin, M., Zhu, K., Caillibotte, G., Noga, M.L., Finlay, W.H., and Martin, A.R. (2017). Comparison of pulsed versus continuous oxygen delivery using realistic adult nasal airway replicas. *Int. J. COPD,* 12:2559–2571.

Churchill, S.E., Shackelford, L.L., Georgi, J.N., and Black, M.T. (2004). Morphological variation and airflow dynamics in the human nose. *Am. J. Hum. Biol.,* 16(6):625–638.

Delvadia, R.R., Longest, P.W., and Byron, P.R. (2012). *In Vitro* Tests for Aerosol Deposition. I: Scaling a Physical Model of the Upper Airways to Predict Drug Deposition Variation in Normal Humans. *J. Aerosol Med. Pulm. Drug Deliv.,* 25(1):32–40.

Djupesland, P.G. (2013). Nasal drug delivery devices: characteristics and performance in a clinical perspective—a review. *Drug Deliv. Transl. Res.,* 3(1):42–62.

Djupesland, P.G. and Skretting, A. (2012). Nasal Deposition and Clearance in Man: Comparison of a Bidirectional Powder Device and a Traditional Liquid Spray Pump. *J. Aerosol Med. Pulm. Drug Deliv.,* 25(5):280–289.

Djupesland, P.G., Skretting, A., Winderen, M., and Holand, T. (2004). Bi-directional nasal delivery of aerosols can prevent lung deposition. *J. Aerosol Med.,* 17(3):249–59.

Garcia, G.J.M., Tewksbury, E.W., Wong, B.A., and Kimbell, J.S. (2009). Interindividual Variability in Nasal Filtration as a Function of Nasal Cavity Geometry. *J. Aerosol Med. Pulm. Drug Deliv.,* 22(2):139–156.

Golshahi, L. and Finlay, W.H. (2012). An Idealized Child Throat that Mimics Average Pediatric Oropharyngeal Deposition. *Aerosol Sci. Technol.,* 46(5):i–iv.

Hahn, I., Scherer, P.W., and Mozell, M.M. (1993). Velocity profiles measured for airflow through a large-scale model of the human nasal cavity. *J. Appl. Physiol.,* 75:2273–2287.

Heyder, J. (2004). Deposition of Inhaled Particles in the Human Respiratory Tract and Consequences for Regional Targeting in Respiratory Drug Delivery. *Proc. Am. Thorac. Soc.,* 1(4):315–320.

Heyder, J. and Rudolf, G. (1975). Deposition of aerosol particles in the human nose. *Inhaled Part,* 4 Pt 1:107–126.

Hounam, R.F., Black, A., and Walsh, M. (1971). The deposition of aerosol particles in the nasopharyngeal region of the human respiratory tract. *J. Aerosol Sci.,* 2(1):47–61.

Hughes, R., Watterson, J., Dickens, C., Ward, D., and Banaszek, A. (2008). Development of a nasal cast model to test medicinal nasal devices. *Proc. Inst. Mech. Eng. Part H-Journal Eng. Med.,* 222(H7):1013–1022.

Javaheri, E., Golshahi, L., and Finlay, W.H. (2013). An idealized geometry that mimics average infant nasal airway deposition. *J. Aerosol Sci.,* 55:137–148.

Kawara, N., Kumita, M., Kurachi, H., Seto, T., Kamba, S., Kondo, T., and Otani, Y. (2016). Sieving of aerosol particles with metal screens. *Aerosol Sci. Technol.,* 50(6):535–541.

Keeler, J.A., Patki, A., Woodard, C.R., and Frank-Ito, D.O. (2015). A Computational Study of Nasal Spray Deposition Pattern in Four Ethnic Groups. *J. Aerosol Med. Pulm. Drug Deliv.,* 28(0):1–14.

Keith, P.K., Desrosiers, M., Laister, T., Schellenberg, R.R., and Waserman, S. (2012). The burden of allergic rhinitis (AR) in Canada: perspectives of physicians and patients. *Allergy Asthma. Clin. Immunol.,* 8(1):7.

Keyhani, K., Scherer, P.W., and Mozell, M.M. (1995). Numerical Simulation of Airflow in the Human Nasal Cavity. *J. Biomech. Eng.,* 117(4):429.

Kiaee, M., Wachtel, H., Noga, M.L., Martin, A.R., and Finlay, W.H. (2018). Regional deposition of nasal sprays in adults: A wide ranging computational study. *Int. j. numer. method. biomed. eng.,* 34(5):e2968.

Kim, S. (2006). Gradient-based simulation optimization.*, in Proceedings - Winter Simulation Conference,* pp. 159–167.

Landahl, H.D. and Black, S. (1947). Penetration of Airborne Particulates through the Human Nose. *J. Ind. Hyg. Toxicol.,* 29(4):269–77.

Leach, C.L., Kuehl, P.J., Chand, R., and McDonald, J.D. (2015). Nasal Deposition of HFA-Beclomethasone, Aqueous Fluticasone Propionate and Aqueous Mometasone Furoate in Allergic Rhinitis Patients. *J. Aerosol Med. Pulm. Drug Deliv.,* 28(5):334–340.

Leach, C.L., Kuehl, P.J., Chand, R., and McDonald, J.D. (2015). Nasal Deposition of HFA-Beclomethasone, Aqueous Fluticasone Propionate and Aqueous Mometasone Furoate in Allergic Rhinitis Patients. *J. Aerosol Med. Pulm. Drug Deliv.,* 28(0):1–7.

Lehrer, S. (2014). Nasal NSAIDs for Alzheimer's Disease. *Am. J. Alzheimer's Dis. Other Dementias®,* 29(5):401–403.

Liu, Y., Johnson, M.R., Matida, E.A., Kherani, S., and Marsan, J. (2009). Creation of a standardized geometry of the human nasal cavity. *J. Appl. Physiol.,* 106(3):784–795.

Liu, Y., Matida, E.A., and Johnson, M.R. (2010). Experimental measurements and computational modeling of aerosol deposition in the Carleton-Civic standardized human nasal cavity. *J. Aerosol Sci.,* 41(6):569–586.

Longest, P.W., Tian, G., Walenga, R.L., and Hindle, M. (2012). Comparing MDI and DPI Aerosol Deposition Using In Vitro Experiments and a New Stochastic Individual Path (SIP) Model of the Conducting Airways. *Pharm. Res.,* 29(6):1670–1688.

Masters, D.A., Poole, D.J., Taylor, N.J., Rendall, T., and Allen, C.B. (2016). Impact of Shape Parameterisation on Aerodynamic Optimisation of Benchmark Problem.*, in 54th AIAA Aerospace Sciences Meeting,* .

Pardeshi, C.V. and Belgamwar, V.S. (2013). Direct nose to brain drug delivery *via* integrated nerve pathways bypassing the blood–brain barrier: an excellent platform for brain targeting. *Expert Opin. Drug Deliv.,* 10(7):957–972.

Patel, R.G., Garcia, G.J.M., Frank-Ito, D.O., Kimbell, J.S., and Rhee, J.S. (2015). Simulating the nasal cycle with computational fluid dynamics. *Otolaryngol. Head. Neck Surg.,* 152(2):353–60.

Pattle, R.E. (1961). The retention of gases and particles in the human nose. *C N Davies, Ed. Inhaled Part. Vapours p,*.

Rhee, J.S., Pawar, S.S., Garcia, G.J.M., and Kimbell, J.S. (2011). Toward Personalized Nasal Surgery Using Computational Fluid Dynamics. *Arch. Facial Plast. Surg.,* 13(5):305.

Rygg, A., Hindle, M., and Longest, P.W. (2016). Linking Suspension Nasal Spray Drug Deposition Patterns to Pharmacokinetic Profiles: A Proof-of-Concept Study Using Computational Fluid Dynamics. *J. Pharm. Sci.,* 105(6):1995–2004.

Schreck, S., Sullivan, K.J., Ho, C.M., and Chang, H.K. (1993). Correlations between flow resistance and geometry in a model of the human nose. *J. Appl. Physiol.,* 75(4):1767–1775.

Schroeter, J.D., Garcia, G.J.M., and Kimbell, J.S. (2010). A computational fluid dynamics approach to assess interhuman variability in hydrogen sulfide nasal dosimetry. *Inhal. Toxicol.,* 22(4):277–286.

Schroeter, J.D., Kimbell, J.S., and Asgharian, B. (2006). Analysis of particle deposition in the turbinate and olfactory regions using a human nasal computational fluid dynamics model. *J. Aerosol Med.,* 19(3):301–313.

Schroeter, J.D., Kimbell, J.S., Asgharian, B., Tewksbury, E.W., and Singal, M. (2012). Computational fluid dynamics simulations of submicrometer and micrometer particle deposition in the nasal passages of a Sprague-Dawley rat. *J. Aerosol Sci.,* 43(1):31–44.

Schroeter, J.D., Tewksbury, E.W., Wong, B.A., and Kimbell, J.S. (2015). Experimental Measurements and Computational Predictions of Regional Particle Deposition in a Sectional Nasal Model. *J. Aerosol Med. Pulm. Drug Deliv.,* 28(1):20–29.

Shah, S. a, Dickens, C.J., Ward, D.J., Banaszek, A. a, George, C., and Horodnik, W. (2013). Design of Experiments to Optimize an In Vitro Cast to Predict Human Nasal Drug Deposition. *J. Aerosol Med. Pulm. Drug Deliv.,* 26(1):1–9.

Song, W. and Keane, A. (2004). A Study of Shape Parameterisation Methods for Airfoil Optimisation., *in 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference,* .

Song, W., Keane, A., Rees, J., Bhaskar, A., and Bagnall, S. (2002). Turbine blade fir-tree root design optimisation using intelligent CAD and finite element analysis. *Comput. Struct.,* 80(24):1853–1867.

Suman, J.D., Laube, B.L., and Dalby, R. (1999). Comparison of nasal deposition and clearance of aerosol generated by a nebulizer and an aqueous spray pump. *Pharm. Res.,* 16(10):1648–1652.

Swift, D.L. (1991). Inspiratory Inertial Deposition of Aerosols in Human Nasal Airway Replicate Casts: Implication for the Proposed NCRP Lung Model. *Radiat. Prot. Dosimetry,* 38(1–3):29–34.

Taylor, D., Radbruch, L., Revnic, J., Torres, L.M., Ellershaw, J.E., and Perelman, M. (2014). A report on the long-term use of fentanyl pectin nasal spray in patients with recurrent breakthrough pain. *J. Pain Symptom Manage.,* 47(6):1001–1007.

Tepper, D.E. (2013). Nasal sprays for the treatment of migraine. *Headache,*.

Tu, J., Inthavong, K., and Ahmadi, G. (2013). The Human Respiratory System. *Comput. Fluid Part. Dyn. Hum. Respir. Syst.,* 3883.

Wang, D.Y., Lee, H.P., and Gordon, B.R. (2012). Impacts of fluid dynamics simulation in study of nasal airflow physiology and pathophysiology in realistic human three-dimensional nose models. *Clin. Exp. Otorhinolaryngol.,* 5(4):181–7.

Warnken, Z.N., Smyth, H.D.C., Watts, A.B., Weitman, S., Kuhn, J.G., and Williams, R.O. (2016). Formulation and device design to increase nose to brain drug delivery. *J. Drug Deliv. Sci. Technol.,*.

Weinhold, I. and Mlynski, G. (2004). Numerical simulation of airflow in the human nose. *Eur. Arch. Oto-Rhino-Laryngology,* 261(8):452–455.

Wen, J., Inthavong, K., Tian, Z., Tu, J., Xue, C., and Li, C. (2007). Airflow patterns in both sides of a realistic human nasal cavity for laminar and turbulent conditions. *16th Australas. Fluid*

Mech. Conf., (December):68–74.

Xi, J., Wang, Z., Nevorski, D., White, T., and Zhou, Y. (2017). Nasal and Olfactory Deposition with Normal and Bidirectional Intranasal Delivery Techniques: *In Vitro* Tests and Numerical Simulations. *J. Aerosol Med. Pulm. Drug Deliv.,* 30(2):118–131.

Xi, J., Yuan, J.E., Zhang, Y., Nevorski, D., Wang, Z., and Zhou, Y. (2016). Visualization and Quantification of Nasal and Olfactory Deposition in a Sectional Adult Nasal Airway Cast. *Pharm. Res.,* 33(6):1527–1541.

Xu, C. and Zhang, J. (2001). A Survey of Quasi-Newton Equations and Quasi-Newton Methods for Optimization. *Ann. Oper. Res.,* 103(1–4):213–234.

Yildiz, A.R., Öztürk, N., Kaya, N., and Öztürk, F. (2003). Integrated optimal topology design and shape optimization using neural networks. *Struct. Multidiscip. Optim.,* 25(4):251–260.

Appendix A: Preprocessing (C++)

```
/*
* -----
* start of opt_manager
* -----
*/

/*
        opt_manager C++ application for postprocessing

        author : milad kiaee darunkola

        Appendix of PhD Thesis

        kiaeedar@ualberta.ca

        -- 2018 --
*/

#ifndef GENNEWPOINTS_POINT_H
#define GENNEWPOINTS_POINT_H

#include <cstdio>
#include <iostream>

class Point {
private:
    std::string flag; // only points with postive flags are considered to be control points
    double x;
    double y;
    double z;
public:
    Point(std::string, double, double, double);
    double getX();
    double getY();
    double getZ();
    std::string getFlag();
    void setFlag(std::string);
    void setX(double);
    void setY(double);
    void setZ(double);
    void print();
};

#endif //GENNEWPOINTS_POINT_H

#include "Point.h"
```

```
Point::Point (std::string f, double xx, double yy, double zz) {
    flag = f;
    x = xx;
    y = yy;
    z = zz;
}

double Point::getX() {
    return x;
}

double Point::getY() {
    return y;
}

double Point::getZ() {
    return z;
}

std::string Point::getFlag() {
    return flag;
}

void Point::setFlag(std::string f) {
    flag = f;
}

void Point::setX(double xx){
    x = xx;
}

void Point::setY(double yy){
    y = yy;
}

void Point::setZ(double zz){
    z = zz;
}

void Point::print(){
    std::cout << "Point: " << x
              << " " << y
              << " " << z << std::endl;
}

/***********************************************************
```

```
#ifndef GENNEWPOINTS_PTSLIS_H
#define GENNEWPOINTS_PTSLIS_H

#include <fstream>
#include <sstream>
#include <utility>
#include <vector>
#include "Point.h"

class PtsLis {
private:
    std::string input_file_name;
    std::string output_file_name;
    std::vector<Point> points;
    size_t n_points;
    //std::vector<BSpline> bsplines;
public:
    PtsLis & operator= (PtsLis);
    void setInputFileName(std::string);
    std::string getInputFileName();
    void setOutputFileName(std::string);
    std::string getOutputFileName();
    size_t getNumPoints();
    void setNumPoints(size_t);
    //std::vector <BSpline> getBSplines();
    std::vector <Point> getPoints();
    void setPoints(std::vector<Point>);
    //void setBSplines(std::vector<BSpline>);
    void differOnePoint(size_t, double);
    void readFile();
    void printFile();
};

#endif //GENNEWPOINTS_PTSLIS_H

#include "PtsLis.h"

PtsLis & PtsLis::operator=(PtsLis A) {
    input_file_name = A.getInputFileName();
    output_file_name = A.getOutputFileName();
    points = A.getPoints();
    n_points = A.getNumPoints();
    return *this;
}

void PtsLis::setPoints(std::vector <Point> cps) {
    points = cps;
```

```
}

void PtsLis::setInputFileName(std::string s) {
    input_file_name = s;
}

std::string PtsLis::getInputFileName() {
    return input_file_name;
}

void PtsLis::setOutputFileName(std::string s) {
    output_file_name = s;
}

std::string PtsLis::getOutputFileName() {
    return output_file_name;
}

size_t PtsLis::getNumPoints() {
    return n_points;
}

void PtsLis::setNumPoints(size_t n) {
    n_points = n;
}

void PtsLis::readFile() {
    std::ifstream myInFile;
    myInFile.open(input_file_name.c_str());
    std::string line;
    std::vector<Point> cps;

    n_points=0;
    if (myInFile.is_open()) {
        while (std::getline(myInFile, line)) {
            //store the lines
            std::stringstream ss(line);
            Point tmp_pt("",0,0,0);
            std::string tmp_f;
            double x, y, z;
            ss >> tmp_f >> x >> y >> z;
            tmp_pt.setX(x);
            tmp_pt.setY(y);
            tmp_pt.setZ(z);
            tmp_pt.setFlag(tmp_f);
            cps.push_back(tmp_pt);
            n_points++;
        }
```

```
    }
    else {
                std::cout << "Error! check if the pts.files can be opened ..."
                  << std::endl;
    }
    n_points = cps.size();
    this->setPoints(cps);
    myInFile.close();
}

void PtsLis::printFile() {
    std::ofstream myOutFile;
    myOutFile.open(output_file_name.c_str());
    for (size_t i=0; i< points.size(); i++) {
        myOutFile << points[i].getFlag() << "  "
                << points[i].getX() << "  "
                << points[i].getY() << "  "
                << points[i].getZ() << std::endl;
    }
    myOutFile.close();
}

std::vector <Point> PtsLis::getPoints() {
    return points;
}

void PtsLis::differOnePoint(size_t i, double dx){
    double xold = points[i].getX();
    points[i].setX(xold + dx);
}

/*************************************************************

#ifndef GENNEWPOINTS_RESULTMATRIX_H
#define GENNEWPOINTS_RESULTMATRIX_H

#include <iostream>
#include <vector>

class ResultMatrix {
private:
    size_t m; // number of rows
    size_t n; // number of columns
    std::vector< std::vector<double> > a;
public:
    ResultMatrix();
```

```
    ResultMatrix(size_t, size_t);
    size_t getM();
    size_t getN();
    void setM(size_t);
    void setN(size_t);
    double getA(size_t, size_t);
    void setA(size_t, size_t, double);
    void addToA(size_t, size_t, double);
    void printA(std::string);
    void devideABy(double);
};

#endif //GENNEWPOINTS_RESULTMATRIX_H

#include "ResultMatrix.h"

ResultMatrix::ResultMatrix() {
    m=5;
    n=4;
    a = std::vector <std::vector <double> > (m, std::vector<double> (n));
}

ResultMatrix::ResultMatrix(size_t mm, size_t nn) {
    m = mm;
    n = nn;
    a = std::vector <std::vector <double> > (m, std::vector<double> (n));
}

size_t ResultMatrix::getM() {
    return m;
}

size_t ResultMatrix::getN() {
    return n;
}

void ResultMatrix::setM(size_t mm) {
    m = mm;
}

void ResultMatrix::setN(size_t nn) {
    n = nn;
}

double ResultMatrix::getA(size_t i, size_t j) {
    return a[i][j];
```

```cpp

void ResultMatrix::setA(size_t i, size_t j, double b) {
    a[i][j] = b;
}

void ResultMatrix::addToA(size_t i, size_t j, double b) {
    a[i][j] += b;
}

void ResultMatrix::printA(std::string name) {
    std::cout << " - - - " << std::endl;
    std::cout << name << " matrix = " << std::endl;
    for (size_t i=0; i<m; i++){
        for (size_t j=0; j<n; j++){
            std::cout << " " << a[i][j];
         }
        std::cout << std::endl;
    }
    std::cout << " - - - " << std::endl;
}

void ResultMatrix::devideABy(double k) {

    for (size_t i=0; i<m; i++){
        for (size_t j=0; j<n; j++){
            a[i][j] = a[i][j]/k;
        }
    }

}

/**********************************************************

/*
 * regions: 1-Vestibule 2-Valve 3-Olfactory 4-Anterior 5-Posterior 6-Naso
 * fractions
 */

#ifndef GENNEWPOINTS_DEPRESULT_H
#define GENNEWPOINTS_DEPRESULT_H

#include "PtsLis.h"
#include "ResultMatrix.h"

class DepResult {
```

```
private:

    // number of position of particle injector tip
    size_t n_injection_points;

    // average contains 6 regional deposition matrices
    std::vector< ResultMatrix > regional_deps;

    std::vector< ResultMatrix > ref_regional_deps_ave;

    // contains 6 regional reference deposition matrices
    std::vector< std::vector< ResultMatrix > > ref_regional_deps;

    std::vector< ResultMatrix > ref_sub1_right_regional_deps;
    std::vector< ResultMatrix > ref_sub2_right_regional_deps;
    std::vector< ResultMatrix > ref_sub3_right_regional_deps;
    std::vector< ResultMatrix > ref_sub4_right_regional_deps;
    std::vector< ResultMatrix > ref_sub6_right_regional_deps;
    std::vector< ResultMatrix > ref_sub7_right_regional_deps;
    std::vector< ResultMatrix > ref_sub8_right_regional_deps;

    std::vector< ResultMatrix > ref_sub1_left_regional_deps;
    std::vector< ResultMatrix > ref_sub2_left_regional_deps;
    std::vector< ResultMatrix > ref_sub3_left_regional_deps;
    std::vector< ResultMatrix > ref_sub4_left_regional_deps;
    std::vector< ResultMatrix > ref_sub6_left_regional_deps;
    std::vector< ResultMatrix > ref_sub7_left_regional_deps;
    std::vector< ResultMatrix > ref_sub8_left_regional_deps;

    double norm;
    double norm_turb;

    double norm_sub1_left;
    double norm_sub2_left;
    double norm_sub3_left;
    double norm_sub4_left;
    double norm_sub6_left;
    double norm_sub7_left;
    double norm_sub8_left;

```

```
    double norm_sub1_right;
    double norm_sub2_right;
    double norm_sub3_right;
    double norm_sub4_right;
    double norm_sub6_right;
    double norm_sub7_right;
    double norm_sub8_right;

    size_t num_particle;
public:
    DepResult();
    DepResult &operator=(DepResult);
    std::vector<ResultMatrix> getRegionalDep();
    std::vector<ResultMatrix> getRefRegionalDep();
    //std::vector<ResultMatrix> getRefRegionalDep_sub4();
    void setNumberOfInjectionPoints(size_t);
    void readFiles(std::string, std::vector<ResultMatrix>&);
    void readRefFiles();
    void findnAdd(std::vector<std::string>, size_t, size_t);
    void readLogFiles(std::string);
    void setNumParticle(size_t);
    size_t getNumParticle();
    void calc_norm();
    void calc_norm_turb();
    void calc_norm_subs();
    void addToNormFile(std::string);
    void addToTurbNormFile(std::string);
    void print_norms();
    void print();
    void test();
    void printRegDepFiles();
    void printMinMaxDepFiles( int );
    void printDepFilesFixVel( int );
};


#endif //GENNEWPOINTS_DEPRESULT_H

#include "DepResult.h"
#include "math.h"
#include <stdlib.h>

DepResult::DepResult() {
    size_t REFS=14;
    size_t N = 6;
    regional_deps = std::vector<ResultMatrix> (N);
    ref_regional_deps_ave = std::vector<ResultMatrix> (N);

```

```
    ref_sub1_right_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub2_right_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub3_right_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub4_right_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub6_right_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub7_right_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub8_right_regional_deps = std::vector< ResultMatrix > (N);

    ref_sub1_left_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub2_left_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub3_left_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub4_left_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub6_left_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub7_left_regional_deps = std::vector< ResultMatrix > (N);
    ref_sub8_left_regional_deps = std::vector< ResultMatrix > (N);

    num_particle = 10000;
    norm = 0;
    norm_turb = 0;

    norm_sub1_left = 0;
    norm_sub2_left = 0;
    norm_sub3_left = 0;
    norm_sub4_left = 0;
    norm_sub6_left = 0;
    norm_sub7_left = 0;
    norm_sub8_left = 0;

    norm_sub1_right = 0;
    norm_sub2_right = 0;
    norm_sub3_right = 0;
    norm_sub4_right = 0;
    norm_sub6_right = 0;
    norm_sub7_right = 0;
```

```
    norm_sub8_right = 0;
}

// read .txt files logs
void DepResult::readFiles(std::string name, std::vector<ResultMatrix>& M) {

    std::vector <std::string> rgs = {"vesti", "valve", "olf", "turbinates",
                                     "naso", "outlet"};
    std::cout << "reading " << name << " files " << std::endl;

    ResultMatrix totalTraced;

    for (size_t m=0; m<5; m++){
        for (size_t n=0; n<4; n++){
                totalTraced.setA(m, n, 0);
        }
    }

    for (size_t i = 0; i < 6; i++){ /* loop over regions */

        std::ifstream f;
        std::string s;
        std::ostringstream oss;
        oss << name.c_str() << rgs[i] << ".txt";
        s = oss.str();

        f.open(s.c_str());

        double x0, x1, x2, x3;
        std::string ln;

        size_t l = 0;

        while (f.is_open() && l<5) {
            getline(f, ln);
            std::stringstream ss(ln);
            ss >> x0 >> x1 >> x2 >> x3;
            x0 *= 100; //change percentage to number of particles out of 10000
            x1 *= 100;
            x2 *= 100;
            x3 *= 100;
            M[i].setA(l, 0, x0);
            M[i].setA(l, 1, x1);
            M[i].setA(l, 2, x2);
```

```
            M[i].setA(l, 3, x3);


            totalTraced.addToA(l, 0, x0);
            totalTraced.addToA(l, 1, x1);
            totalTraced.addToA(l, 2, x2);
            totalTraced.addToA(l, 3, x3);

            l++;
        }
        f.close();
    }

    for (size_t i=0; i<6; i++){

        for (size_t m=0; m<5; m++){
                for (size_t n=0; n<4; n++){

                        //totalTraced.getA(m, n);
                        double tmp = M[i].getA(m, n)/10000.0;
                        std::cout << "region " << rgs[i] <<
                        " index [" << m << ", " << n  <<
                        "] of ref deviding by total " << tmp << std::endl;
                        M[i].setA(m, n, tmp);
                        //M[i].printA("refread_update");
                }
        }

    }
}

void DepResult::readRefFiles(){
    std::cout << "###################### " << std::endl;
    readFiles("ref_", ref_regional_deps_ave);

        readFiles("ref_sub1_right_", ref_sub1_right_regional_deps);
        readFiles("ref_sub2_right_", ref_sub2_right_regional_deps);
        readFiles("ref_sub3_right_", ref_sub3_right_regional_deps);
        readFiles("ref_sub4_right_", ref_sub4_right_regional_deps);
        readFiles("ref_sub6_right_", ref_sub6_right_regional_deps);
```

```
        readFiles("ref_sub7_right_", ref_sub7_right_regional_deps);
        readFiles("ref_sub8_right_", ref_sub8_right_regional_deps);
        readFiles("ref_sub1_left_", ref_sub1_left_regional_deps);
        readFiles("ref_sub2_left_", ref_sub2_left_regional_deps);
        readFiles("ref_sub3_left_", ref_sub3_left_regional_deps);
        readFiles("ref_sub4_left_", ref_sub4_left_regional_deps);
        readFiles("ref_sub6_left_", ref_sub6_left_regional_deps);
        readFiles("ref_sub7_left_", ref_sub7_left_regional_deps);
        readFiles("ref_sub8_left_", ref_sub8_left_regional_deps);

        ref_regional_deps.push_back(ref_sub1_right_regional_deps);
        ref_regional_deps.push_back(ref_sub2_right_regional_deps);
        ref_regional_deps.push_back(ref_sub3_right_regional_deps);
        ref_regional_deps.push_back(ref_sub4_right_regional_deps);
        ref_regional_deps.push_back(ref_sub6_right_regional_deps);
        ref_regional_deps.push_back(ref_sub7_right_regional_deps);
        ref_regional_deps.push_back(ref_sub8_right_regional_deps);
        ref_regional_deps.push_back(ref_sub1_left_regional_deps);
        ref_regional_deps.push_back(ref_sub2_left_regional_deps);
        ref_regional_deps.push_back(ref_sub3_left_regional_deps);
        ref_regional_deps.push_back(ref_sub4_left_regional_deps);
        ref_regional_deps.push_back(ref_sub6_left_regional_deps);
        ref_regional_deps.push_back(ref_sub7_left_regional_deps);
        ref_regional_deps.push_back(ref_sub8_left_regional_deps);

```

```
/*
        for (size_t m=0; m<5; m++){
                for(size_t n=0; n<4; n++){
                        for (size_t region=0; region<6; region++){
                                for (size_t ref=0; ref<14; ref++){
                                        ref_regional_deps_ave[
                                        region
                                        ].addToA(
                                        m, n,
                                ref_regional_deps.at(ref).at(region).getA(m, n)
                                        );
                                }
                        }
                }
        }

        for (size_t region=0; region<6; region++){
                ref_regional_deps_ave[region].devideABy(14);
        }
*/
    std::cout << "######################" << std::endl;
}

void DepResult::findnAdd(std::vector<std::string> SV, size_t m, size_t n) {

    int v;
    char c;
    std::string S;
    std::vector <std::string> rgs = {"VESTIBULE", "VALVE", "OLF",
                                     "TURBINATES", "RODS", "OBS",
                                     "NASO", "OUTLET"};
    std::vector <std::string> actualRgs = {"VESTIBULE", "VALVE", "OLF",
                                           "TURBINATES", "NASO", "OUTLET"};
    int index=0;

```

```
    /* for each region loop through deposition log file */

    double totalTraced = 0;

    std::vector<ResultMatrix> tmpResult;

    for (size_t i=0; i<6; i++){
        ResultMatrix tmpMat;
        tmpResult.push_back(tmpMat);
        for (size_t m=0; m<5; m++){
            for (size_t n=0; n<4; n++){
                tmpResult[i].setA(m, n, 0);
                if (i == 5){
                        tmpResult[i].setA(m, n, 10000);
                }
            }
        }
    }

    for (int j = 0; j < 8; j++) { // regions
        for (int k = SV.size() - 1; k > 0; k--) { // lines

            if (SV[k].find(rgs[j].c_str()) !=
std::string::npos) {

                int ind = 0;

                if ( j == 7 ) {
                        ind = k + 1;
                } else {
                        ind = k + 2;
                }

                std::stringstream ss(SV[ind]);
                ss >> c >> S >> c >> S;
                std::stringstream ss2(S);
                std::getline(ss2, S, ',');
                v = atoi(S.c_str());

                 /* set to 10,000 ususally, varies
                    based of simulation (set in main) */
                double dn = double(num_particle);

                double dv = double(v);

                totalTraced += dv;

                /* only be used if other normalization is
```

```
not used */
                //dv = double(v) / dn;

                if (j == 4 || j == 5) {
                  index --;
                }

                double tempValue = -1 * dv;

                if (index != 5) {

                        tmpResult[5].addToA(m, n, tempValue);

                         /* adds the stick counts to regdep matrix */
                        tmpResult[index].addToA(m, n, dv);

                }

                index ++;
                break;
            }
        }
    }

    // this should go to the superfunction
    /* only be used if no other normalization
        is used in this or the caller function */
    for (int j = 0; j < 6; j++) {
        //std::cout << "deviding by total traced " << totalTraced << std::endl;
        double tmp = tmpResult[j].getA(m, n) / 10000.0;// double(totalTraced);

        regional_deps[j].addToA(m, n, tmp);

        //std::cout << "matrix " << actualRgs[j] << std::endl;
        regional_deps[j].printA("log_update");
    }
}

void DepResult::readLogFiles(std::string logName) {

    std::vector<double> u0 = {0, 5, 10, 20};
    std::vector<double> diam = {5e-06, 1e-05, 1.5e-05, 2e-05, 4e-05};
```

```
    std::ifstream f;
    //std::cout << "reading log files .." << std::endl;

    for (size_t i=1; i<=n_injection_points; i++) {

        for (size_t m=0; m<5; m++) {
            for (size_t n=0; n<4; n++) {

                std::string ln, fileName;
                std::vector<std::string> lns;
                std::ostringstream oss;
                oss << "_" << i << "_" << diam[m] << "_" << u0[n];
                fileName = logName + oss.str();
                f.open(fileName.c_str());

                if (f.is_open())
                    while (std::getline(f, ln))
                        lns.push_back(ln);

                //std::cout << "file " << fileName << " ... " << std::endl;

                findnAdd(lns, m, n);
                f.close();

            }
        }
    }

    // calculate fraction over all injection positions
    for (int j = 0; j < 6; j++) {
        std::cout << "deviding by " << n_injection_points << std::endl;
        regional_deps[j].devideABy(double(n_injection_points));
    }


    std::cout << "######################" << std::endl;
}

void DepResult::calc_norm() {
    size_t m = regional_deps[0].getM();
    size_t n = regional_deps[0].getN();

    //homogeneous scalarization using ferobenous as objective function
```

```
    for (size_t q=0; q<regional_deps.size(); q++) {
        for (size_t i = 0; i < m; i++) {
            for (size_t j = 0; j < n; j++) {
                norm += pow(regional_deps[q].getA(i, j)
                        - ref_regional_deps_ave[q].getA(i,
j), 2);
            }
        }
    }
    norm = sqrt(norm);
    norm = norm / 6 / ( 4 * 5 );
}

void DepResult::calc_norm_turb() {
    size_t m = regional_deps[0].getM();
    size_t n = regional_deps[0].getN();
    //homogeneous scalarization using ferobenous as
objective function
    size_t q = 3;
    for (size_t i = 0; i < m; i++) {
      for (size_t j = 0; j < n; j++) {
        norm_turb += pow(regional_deps[q].getA(i, j)
                        - ref_regional_deps_ave[q].getA(i,
j), 2);
      }
    }
    norm_turb = sqrt(norm_turb);
    norm_turb = norm_turb / ( 4 * 5 );
}

void DepResult::calc_norm_subs() {
    size_t m = regional_deps[0].getM();
    size_t n = regional_deps[0].getN();

    //homogeneous scalarization using ferobenous as
objective function
    for (size_t q=0; q<regional_deps.size(); q++) {
        for (size_t i = 0; i < m; i++) {
            for (size_t j = 0; j < n; j++) {
                norm_sub1_right += pow
(ref_sub1_right_regional_deps[q].getA(i, j)
                                        -
ref_regional_deps_ave[q].getA(i, j), 2);
                norm_sub2_right += pow
(ref_sub2_right_regional_deps[q].getA(i, j)
                                        -
ref_regional_deps_ave[q].getA(i, j), 2);
                norm_sub3_right += pow
```

```
(ref_sub3_right_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub4_right += pow (ref_sub4_right_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub6_right += pow (ref_sub6_right_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub7_right += pow (ref_sub7_right_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub8_right += pow (ref_sub8_right_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);

                    norm_sub1_left += pow (ref_sub1_left_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub2_left += pow (ref_sub2_left_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub3_left += pow (ref_sub3_left_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub4_left += pow (ref_sub4_left_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub6_left += pow (ref_sub6_left_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub7_left += pow (ref_sub7_left_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
                    norm_sub8_left += pow (ref_sub8_left_regional_deps[q].getA(i, j)
                                                  - ref_regional_deps_ave[q].getA(i, j), 2);
```

```
            }
        }
    }
    norm_sub1_right = sqrt(norm_sub1_right);
    norm_sub1_right = norm_sub1_right / 6 / (4*5);
    norm_sub2_right = sqrt(norm_sub2_right);
    norm_sub2_right = norm_sub2_right / 6 / (4*5);
    norm_sub3_right = sqrt(norm_sub3_right);
    norm_sub3_right = norm_sub3_right / 6 / (4*5);
    norm_sub4_right = sqrt(norm_sub4_right);
    norm_sub4_right = norm_sub4_right / 6 / (4*5);
    norm_sub6_right = sqrt(norm_sub6_right);
    norm_sub6_right = norm_sub6_right / 6 / (4*5);
    norm_sub7_right = sqrt(norm_sub7_right);
    norm_sub7_right = norm_sub7_right / 6 / (4*5);
    norm_sub8_right = sqrt(norm_sub8_right);
    norm_sub8_right = norm_sub8_right / 6 / (4*5);

    norm_sub1_left = sqrt(norm_sub1_left);
    norm_sub1_left = norm_sub1_left / 6 / (4*5);
    norm_sub2_left = sqrt(norm_sub2_left);
    norm_sub2_left = norm_sub2_left / 6 / (4*5);
    norm_sub3_left = sqrt(norm_sub3_left);
    norm_sub3_left = norm_sub3_left / 6 / (4*5);
    norm_sub4_left = sqrt(norm_sub4_left);
    norm_sub4_left = norm_sub4_left / 6 / (4*5);
    norm_sub6_left = sqrt(norm_sub6_left);
    norm_sub6_left = norm_sub6_left / 6 / (4*5);
    norm_sub7_left = sqrt(norm_sub7_left);
    norm_sub7_left = norm_sub7_left / 6 / (4*5);
    norm_sub8_left = sqrt(norm_sub8_left);
    norm_sub8_left = norm_sub8_left / 6 / (4*5);
}

void DepResult::addToNormFile( std::string K) {

    // write down to the result.out file which is for the dakota code
    std::ofstream dakf; //dakota file
    dakf.open(K.c_str(), std::fstream::app);
    calc_norm();
    std::ostringstream oss;
    oss << "norm";
    std::string s = oss.str();
    dakf << norm << " " << s << std::endl;
    dakf.close();

    // write appending norm.tmp file for future analysis
```

```
    std::ofstream f;
    f.open("norm.tmp", std::fstream::app);
    f << norm << std::endl;
    f.close();
}

void DepResult::addToTurbNormFile( std::string K) {

    // write down to the result.out file which is for the dakota code
    std::ofstream dakf; //dakota file
    dakf.open(K.c_str(), std::fstream::app);
    calc_norm_turb();
    std::ostringstream oss;
    oss << "norm";
    std::string s = oss.str();
    dakf << norm_turb << " " << s << std::endl;
    dakf.close();

    // write appending norm.tmp file for future analysis
    std::ofstream f;
    f.open("norm.tmp", std::fstream::app);
    f << norm_turb << std::endl;
    f.close();
}

void DepResult::print_norms(){
    std::cout << "----------" << std::endl;
    calc_norm_subs();

    std::cout << "Current geometry error is <<" << 100*norm
              << " %>> of total <<" << num_particle
              << ">> particles" << std::endl;
    std::cout << "Subject 1 left error is <<" << 100*norm_sub1_left
              << " %>> of total <<" << num_particle
              << ">> praticles" << std::endl;
    std::cout << "Subject 2 left error is <<" << 100*norm_sub2_left
              << " %>> of total <<" << num_particle
              << ">> praticles" << std::endl;
    std::cout << "Subject 3 left error is <<" << 100*norm_sub3_left
              << " %>> of total <<" << num_particle
              << ">> praticles" << std::endl;
    std::cout << "Subject 4 left error is <<" << 100*norm_sub4_left
              << " %>> of total <<" << num_particle
```

```
                << ">> praticles" << std::endl;
    std::cout << "Subject 6 left error is <<" << 100*norm_sub6_left
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;
    std::cout << "Subject 7 left error is <<" << 100*norm_sub7_left
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;
    std::cout << "Subject 8 left error is <<" << 100*norm_sub8_left
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;

    std::cout << "Subject 1 right error is <<" << 100*norm_sub1_right
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;
    std::cout << "Subject 2 right error is <<" << 100*norm_sub2_right
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;
    std::cout << "Subject 3 right error is <<" << 100*norm_sub3_right
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;
    std::cout << "Subject 4 right error is <<" << 100*norm_sub4_right
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;
    std::cout << "Subject 6 right error is <<" << 100*norm_sub6_right
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;
    std::cout << "Subject 7 right error is <<" << 100*norm_sub7_right
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;
    std::cout << "Subject 8 right error is <<" << 100*norm_sub8_right
                << " %>> of total <<" << num_particle
                << ">> praticles" << std::endl;

    std::cout << "----------" << std::endl;
}

void DepResult::test(){

```

```
    std::cout << "this is a test: " << std::endl;
    std::string s = "ref sub4";

    for (size_t i=0; i<6; i++)
        ref_sub4_left_regional_deps[i].printA(s);

    s = "ref";

    for (size_t i=0; i<6; i++)
        ref_regional_deps_ave[i].printA(s);

    std::cout << "end of test" << std::endl;
}

void DepResult::printRegDepFiles(){
    std::ofstream f;
    std::vector <std::string> rgs = {"vesti.txt", "valve.txt", "olf.txt",
                                     "turbinates.txt", "naso.txt",
                                     "outlet.txt"};
    for (size_t i=0; i<6; i++){
        regional_deps[i].printA("Average deposition in " + rgs[i]);
        f.open(rgs[i].c_str());
        for (size_t m=0; m<5; m++){
            for (size_t n=0; n<4; n++){
                f << regional_deps[i].getA(m, n) << " ";
            }
            f << std::endl;
        }
        f.close();
    }
}

void DepResult::printMinMaxDepFiles(int velIndex){
        // prints a file for a specific velocity with this format:
        // diameter deposition min max
        std::ofstream f;
        std::vector <std::string> rgs = {"vestiEr.txt", "valveEr.txt",
                                         "olfEr.txt" , "turbinatesEr.txt",
                                         "nasoEr.txt", "outletEr.txt"};

        for (size_t region=0; region<6; region++){
```

```
        std::string s(rgs[region]);
        f.open(s.c_str());
        f << "0 0 0 0" << std::endl;

        for (size_t diameter=0; diameter<5; diameter++){

            double min = ref_regional_deps.at(0).at(region).
                                    getA(diameter, velIndex);
            double max = ref_regional_deps.at(0).at(region).
                                    getA(diameter, velIndex);

            for (size_t ref=1; ref<14; ref++){
                double tmp = ref_regional_deps.at(ref).
                                at(region).getA(diameter, velIndex) ;

                if ( min > tmp ) {
                    min = tmp;
                }
                if ( max < tmp) {
                    max = tmp;
                }

            }

            f << diameter+1 << " " << regional_deps[region].
                getA(diameter, velIndex)
                << " " << min << " " << max;
            f << std::endl;
        }

        f.close();
    }
}

void DepResult::printDepFilesFixVel(int velIndex){
    // prints a file for a specific velocity with this format:
    // diameter deposition min max
    std::ofstream f;
```

```
        std::vector <std::string> rgs = { "vesti", "valve", "olf" ,
                                          "turbinates", "naso", "outlet" };
        std::vector<double> diam = { 5e-06, 1e-05, 1.5e-05, 2e-05, 4e-05 };
        std::vector<double> diam_micron = { 5, 10, 15, 20, 40 };

        for (size_t region=0; region<6; region++){

                std::string s = rgs[region] ;
                s.append("vel");
                s.append(std::to_string(velIndex));
                s.append(".txt");
                f.open(s.c_str());
                //f << "" << std::endl;

                for (size_t diameter=0; diameter<5; diameter++){

                        for (size_t ref=0; ref<14; ref++){
                                double tmp = ref_regional_deps.at(ref).
                                        at(region).getA(diameter, velIndex) ;
                                f << diam_micron[diameter] << " " << tmp
                                        << " 1" << std::endl;

                        }

                        f << diam_micron[diameter] << " "
                                << regional_deps[region].getA(diameter, velIndex)
                                << " 2" << std::endl;
                        f << std::endl;
                }

                f.close();
        }
}

std::vector <ResultMatrix> DepResult::getRegionalDep() {
    return regional_deps;
}
```

```
std::vector <ResultMatrix> DepResult::getRefRegionalDep() {
    return ref_regional_deps_ave;
}

void DepResult::setNumberOfInjectionPoints(size_t n) {
    n_injection_points = n;
}

void DepResult::setNumParticle(size_t n) {
    num_particle = n;
}

size_t DepResult::getNumParticle() {
    return num_particle;
}

/************************************************************

#include "PtsLis.h"
#include "DepResult.h"
#include <cmath>

int main(int argc, char* argv[]) {

    std::string input = argv[1];
    if (input == "append_norm")
    {
        std::cout << "appending norm" << std::endl;
        std::string s = argv[2];
        DepResult d;
        size_t numPoints = 200;
        d.setNumberOfInjectionPoints(numPoints);

        std::cout << "Reading Reference Files .. " << std::endl;
        d.readRefFiles(); //reads the reference values
        std::cout << "Reading Reference Files Done" << std::endl;

        std::cout << "Reading Log Files .. " << std::endl;
        d.readLogFiles("plog");
        std::cout << "Reading Log Files Done" << std::endl;

        d.addToNormFile(s);
        d.print_norms();
        d.printRegDepFiles();
```

```

        d.printMinMaxDepFiles(0); //velocity index

        d.printDepFilesFixVel(0); // print all vel based dep files
        d.printDepFilesFixVel(2);
        d.printDepFilesFixVel(3);
    }

    else if (input == "append_norm_turb")
    {
        std::cout << "appending norm" << std::endl;
        std::string s = argv[2];
        DepResult d;
        size_t numPoints = 4;
        d.setNumberOfInjectionPoints(numPoints);
        d.readRefFiles(); //reads the reference values
        d.readLogFiles("plog");
        d.addToNormFile(s);
        //      d.print_norms();
        d.printRegDepFiles();
       // d.test();
    } else {
      std::cout << "unknown input argument." << std::endl;
    }

    return 0;
}

/*
* -----
* end of opt_manager
* -----
*/
```

Appendix B: Visualization Toolkit (VTK)

```
/**
** start of VTK codes
**/
/*
        Several VTK codes for different functions
        to execute choose a main method and include headers
        to create a suitable object. Operations include:
        closeClip, closeAtFeature, fillHoles, flipNormals
        pointsInside, girdSTL, puncher, scaleSTL

        version of code: 3.0

        Author: Milad Kiaee Darunkola kiaeedar@ualberta.ca
        Appendix to Thesis

*/

#include <vtkVersion.h>
#include <vtkSmartPointer.h>

#include <vtkClipDataSet.h>
#include <vtkImplicitPolyDataDistance.h>
#include <vtkConeSource.h>
#include <vtkPointData.h>
#include <vtkUnstructuredGrid.h>
#include <vtkFloatArray.h>
#include <vtkRectilinearGrid.h>
#include <vtkPolyDataMapper.h>
#include <vtkProperty.h>
#include <vtkActor.h>
#include <vtkCamera.h>
#include <vtkRectilinearGridGeometryFilter.h>
#include <vtkDataSetMapper.h>
#include <vtkRenderer.h>
#include <vtkRenderWindow.h>
#include <vtkRenderWindowInteractor.h>
#include <vtkSTLReader.h>
#include <vtkSTLWriter.h>
#include <vtkXMLPolyDataReader.h>
#include <vtkXMLPolyDataWriter.h>
#include <vtkPLYWriter.h>
#include <vtkPolyDataWriter.h>
#include <vtkDataSetWriter.h>
#include <vtkUnstructuredGridGeometryFilter.h>
#include <vtkDataSetSurfaceFilter.h>
#include <vtkCubeSource.h>
#include <vtkSphereSource.h>
#include <vtkTableBasedClipDataSet.h>
```

```
#include <vtkTriangleFilter.h>
#include <map>

#include <sstream>
#include <vector>
#include <stdlib.h> /* srand, rand */
#include <time.h> /* time */

#include <vtkCleanPolyData.h>
#include <vtkAppendPolyData.h>
#include <vtkAppendFilter.h>
#include <vtkDelaunay2D.h>
#include <vtkConnectivityFilter.h>
#include <vtkPolyDataConnectivityFilter.h>
#include <vtkSelectionNode.h>
#include <vtkInformation.h>
#include <vtkFillHolesFilter.h>

#include <vtkTransformPolyDataFilter.h>
#include <vtkTransform.h>
#include <vtkTransformPolyDataFilter.h>


int main (int argc, char *argv[])
{
  std::cout << "argc = " << argc << std::endl;
  // Create polydata to slice the grid with.
  // In this case, use a cone. This could
  // be any polydata including a stl file.

  // PolyData to process
  std::string input_name1(argv[1]);
  std::cout << "Reading stl file : " << input_name1 << std::endl;
  vtkSmartPointer<vtkSTLReader> stlReader1 =
    vtkSmartPointer<vtkSTLReader>::New();
  stlReader1->SetFileName(input_name1.c_str());
  stlReader1->Update();
  vtkSmartPointer<vtkPolyData> pd1;
  pd1 = stlReader1->GetOutput();

  // Implicit function that will be used to slice the mesh
  vtkSmartPointer<vtkImplicitPolyDataDistance> implicitPolyDataDistance =
    vtkSmartPointer<vtkImplicitPolyDataDistance>::New();
  implicitPolyDataDistance->SetInput(pd1);

  // PolyData to process
```

```
  std::string input_name2(argv[2]);
  std::cout << "Reading stl file : " << input_name2 << std::endl;
  vtkSmartPointer<vtkSTLReader> stlReader2 =
    vtkSmartPointer<vtkSTLReader>::New();
  stlReader2->SetFileName(input_name2.c_str());
  stlReader2->Update();
  vtkSmartPointer<vtkPolyData> pd2;
  pd2 = stlReader2->GetOutput();

  // Create an array to hold distance information
  vtkSmartPointer<vtkFloatArray> signedDistances =
    vtkSmartPointer<vtkFloatArray>::New();
  signedDistances->SetNumberOfComponents(1);
  signedDistances->SetName("SignedDistances");

  double extra = -0.0005;

  if ( argc  > 4){
    extra = 0.0005;
  }

  // Evaluate the signed distance function at all of the grid points
  for (vtkIdType pointId = 0; pointId < pd2->GetNumberOfPoints(); ++pointId)
  {
    double p[3];
    pd2->GetPoint(pointId, p);
    double signedDistance = implicitPolyDataDistance->EvaluateFunction(p) + extra;
    signedDistances->InsertNextValue(signedDistance);
  }

  // Add the SignedDistances to the grid
  pd2->GetPointData()->SetScalars(signedDistances);

  // Use vtkClipDataSet to slice the grid with the polydata
  vtkSmartPointer<vtkTableBasedClipDataSet> clipper =
    vtkSmartPointer<vtkTableBasedClipDataSet>::New();

  clipper->SetInputData(pd2);
  if ( argc  > 4){
    std::cout << "InsideOut is ON." << std::endl;
    clipper->InsideOutOn();
  }
  clipper->SetValue(0.00);
  //clipper->SetOutputPointsPrecision(20);
```

```
  clipper->GenerateClippedOutputOn();
  clipper->Update();

/*
  vtkSmartPointer<vtkUnstructuredGridGeometryFilter> uggf =
    vtkSmartPointer<vtkUnstructuredGridGeometryFilter>::New();
  uggf->SetInputData(clipper->GetOutput());
  uggf->Update();
*/

  vtkSmartPointer<vtkDataSetSurfaceFilter> dssf =
    vtkSmartPointer<vtkDataSetSurfaceFilter>::New();
  dssf->SetInputData(clipper->GetOutput());
  dssf->Update();

  vtkSmartPointer<vtkTriangleFilter> tf =
    vtkSmartPointer<vtkTriangleFilter>::New();
  tf->SetInputData(dssf->GetOutput());
  tf->Update();

  std::string outname(argv[3]);

/*
  std::string outPly = outname + ".ply";
  // write the detected boundary edges
  vtkSmartPointer<vtkXMLPolyDataWriter> writer
        = vtkSmartPointer<vtkXMLPolyDataWriter>::New();
  writer->SetInputConnection(tf->GetOutputPort());
  writer->SetFileName(outPly.c_str());
  writer->Write();
*/

  std::string outSTL = outname;
  vtkSmartPointer<vtkSTLWriter> sw2
        = vtkSmartPointer<vtkSTLWriter>::New();
  sw2->SetFileName(outSTL.c_str());
  std::cout << "writing stl .. " << std::endl;
  sw2->SetInputData(tf->GetOutput());
  sw2->Write();

  // ********************************************************
  // Uncomment to Generate a report
/*
  vtkIdType numberOfCells = clipper->GetOutput()->GetNumberOfCells();
  std::cout << "------------------------" << std::endl;
  std::cout << "The clipped dataset(inside) contains a " <<
```

```
std::endl
            << clipper->GetOutput()->GetClassName()
            <<  " that has " << numberOfCells << " cells" << std::endl;
  typedef std::map<int,int> CellContainer;
  CellContainer cellMap;
  for (vtkIdType i = 0; i < numberOfCells; i++)
  {
    cellMap[clipper->GetOutput()->GetCellType(i)]++;
  }

  CellContainer::const_iterator it = cellMap.begin();
  while (it != cellMap.end())
  {
    std::cout << "\tCell type "
              << vtkCellTypes::GetClassNameFromTypeId(it->first)
              << " occurs " << it->second << " times." << std::endl;
    ++it;
  }

  numberOfCells = clipper->GetClippedOutput()->GetNumberOfCells();
  std::cout << "------------------------" << std::endl;
  std::cout << "The clipped dataset(outside) contains a " << std::endl
            << clipper->GetClippedOutput()->GetClassName()
            <<  " that has " << numberOfCells << " cells" << std::endl;
  typedef std::map<int,int> OutsideCellContainer;
  CellContainer outsideCellMap;
  for (vtkIdType i = 0; i < numberOfCells; i++)
  {
    outsideCellMap[clipper->GetClippedOutput()->GetCellType(i)]++;
  }

  it = outsideCellMap.begin();
  while (it != outsideCellMap.end())
  {
    std::cout << "\tCell type "
              << vtkCellTypes::GetClassNameFromTypeId(it->first)
              << " occurs " << it->second << " times." << std::endl;
    ++it;
  }
```

```
*/

  return EXIT_SUCCESS;
}

int main (int argc, char *argv[])
{
  // PolyData to process
  std::string inputName1(argv[1]);
  std::cout << "Reading stl file : " << inputName1 <<
std::endl;
  vtkSmartPointer<vtkSTLReader> stlReader1 =
    vtkSmartPointer<vtkSTLReader>::New();
  stlReader1->SetFileName(inputName1.c_str());
  stlReader1->Update();
  vtkSmartPointer<vtkPolyData> polyData1;
  polyData1 = stlReader1->GetOutput();

  // PolyData to process
  std::string inputName2(argv[2]);
  std::cout << "Reading stl file : " << inputName2 <<
std::endl;
  vtkSmartPointer<vtkSTLReader> stlReader2 =
    vtkSmartPointer<vtkSTLReader>::New();
  stlReader2->SetFileName(inputName2.c_str());
  stlReader2->Update();
  vtkSmartPointer<vtkPolyData> polyData2;
  polyData2 = stlReader2->GetOutput();

// ********
        vtkSmartPointer<vtkFeatureEdges> boundaryEdges1 =
                vtkSmartPointer<vtkFeatureEdges>::New();
        boundaryEdges1->SetInputData(polyData1);
        boundaryEdges1->BoundaryEdgesOn();
        boundaryEdges1->FeatureEdgesOff();
        boundaryEdges1->NonManifoldEdgesOff();
        boundaryEdges1->ColoringOff();
        boundaryEdges1->Update();

        vtkSmartPointer<vtkFeatureEdges> boundaryEdges2 =
                vtkSmartPointer<vtkFeatureEdges>::New();
        boundaryEdges2->SetInputData(polyData2);
        boundaryEdges2->BoundaryEdgesOn();
        boundaryEdges2->FeatureEdgesOff();
        boundaryEdges2->NonManifoldEdgesOff();
        boundaryEdges2->ColoringOff();
        boundaryEdges2->Update();

```

```
// ********

        vtkSmartPointer<vtkPolyDataConnectivityFilter> connectivityFilter1 =
                vtkSmartPointer<vtkPolyDataConnectivityFilter>::New();
        connectivityFilter1->SetInputData(boundaryEdges1->GetOutput());
        connectivityFilter1->SetExtractionModeToSpecifiedRegions();
        connectivityFilter1->AddSpecifiedRegion(0);
        connectivityFilter1->Update();



        vtkSmartPointer<vtkPolyDataConnectivityFilter> connectivityFilter2 =
                vtkSmartPointer<vtkPolyDataConnectivityFilter>::New();
        connectivityFilter2->SetInputData(boundaryEdges2->GetOutput());
        connectivityFilter2->SetExtractionModeToSpecifiedRegions();
        connectivityFilter2->AddSpecifiedRegion(0);
        connectivityFilter2->Update();

// *********

        vtkSmartPointer<vtkCleanPolyData> cleanPolyData1 =
                vtkSmartPointer<vtkCleanPolyData>::New();
        cleanPolyData1->SetInputData(connectivityFilter1->GetOutput());
        cleanPolyData1->Update();

            // Write the file
  vtkSmartPointer<vtkXMLPolyDataWriter> writer1 =
    vtkSmartPointer<vtkXMLPolyDataWriter>::New();
  writer1->SetFileName("test1.vtp");
  writer1->SetInputData(cleanPolyData1->GetOutput());
  // Optional - set the mode. The default is binary.
  //writer->SetDataModeToBinary();
  //writer->SetDataModeToAscii();
  writer1->Write();

        vtkSmartPointer<vtkCleanPolyData> cleanPolyData2 =
                vtkSmartPointer<vtkCleanPolyData>::New();
        cleanPolyData2->SetInputData(connectivityFilter2->GetOutput());
```

```
        cleanPolyData2->Update();

            // Write the file
  vtkSmartPointer<vtkXMLPolyDataWriter> writer2 =
    vtkSmartPointer<vtkXMLPolyDataWriter>::New();
  writer2->SetFileName("test2.vtp");
  writer2->SetInputData(cleanPolyData2->GetOutput());
  // Optional - set the mode. The default is binary.
  //writer->SetDataModeToBinary();
  //writer->SetDataModeToAscii();
  writer2->Write();

// ********

        vtkSmartPointer<vtkAppendPolyData> appendPolyDataFilter =
            vtkSmartPointer<vtkAppendPolyData>::New();
        appendPolyDataFilter->AddInputData( cleanPolyData1->GetOutput() );
        appendPolyDataFilter->AddInputData( cleanPolyData2->GetOutput() );
        appendPolyDataFilter->Update();

        // ********

        vtkSmartPointer<vtkDelaunay2D> delauny =
            vtkSmartPointer<vtkDelaunay2D>::New();
        delauny->SetInputData(appendPolyDataFilter->GetOutput());
        delauny->SetProjectionPlaneMode(VTK_BEST_FITTING_PLANE);
        delauny->Update();

/*
        std::ostringstream ss;
        std::string out (argv[1]);
        ss << out << ".stl";
        out  = ss.str();

        std::string name1(out);
        vtkSmartPointer<vtkSTLWriter> writer =
                vtkSmartPointer<vtkSTLWriter>::New();
        writer1->SetFileName(name1.c_str());
        std::cout << "writing .. " << std::endl;
        writer->SetInputData(delauny->GetOutput());
        writer->Write();

*/
```

```
  return EXIT_SUCCESS;
}

int main(int argc, char *argv[])
{
  // defaults to be changed
  std::string input_name(argv[1]);
  std::cout << "filling holes of : " << input_name << std::endl;
  // read two stls
  vtkSmartPointer<vtkSTLReader> sr = vtkSmartPointer<vtkSTLReader>::New();
  sr->SetFileName(input_name.c_str());
  sr->Update();;
  // store then in polydata files
  vtkSmartPointer<vtkPolyData> input;
  input = sr->GetOutput(); //or try shallowcopy

  vtkSmartPointer<vtkFillHolesFilter> fhf = vtkSmartPointer<vtkFillHolesFilter>::New();
  fhf->SetInputData(input);

  fhf->SetHoleSize(0.1);

  // Make the triangle windong order consistent
  vtkSmartPointer<vtkPolyDataNormals> normals = vtkSmartPointer<vtkPolyDataNormals>::New();
  normals->SetInputConnection(fhf->GetOutputPort());
  normals->ConsistencyOn();
  normals->SplittingOff();
  normals->Update();

  // Restore the original normals
  normals->GetOutput()->GetPointData()->SetNormals(input->GetPointData()->GetNormals());

  vtkSmartPointer<vtkDataSetSurfaceFilter> sf = vtkSmartPointer<vtkDataSetSurfaceFilter>::New();
  sf->SetInputConnection(fhf->GetOutputPort());
  sf->Update();

  // stl writer
  std::cout << "fill holes: stl writer starting .. " << std::endl;
  vtkSmartPointer<vtkSTLWriter> sw = vtkSmartPointer<vtkSTLWriter>::New();
  sw->SetFileName(argv[2]);
  sw->SetInputConnection(sf->GetOutputPort());
```

```
  sw->SetFileTypeToBinary();
  std::cout << "fill holes: writing .. " << std::endl;
  sw->Write();

  return EXIT_SUCCESS;
}

int main ( int argc, char *argv[] )
{
  std::cout << "grid std: usuage: ./obj input ox oy oz dy dz ny nz" << std::endl;

  std::string input_name_1 = argv[1];

  vtkSmartPointer<vtkSTLReader> sr_1 =
    vtkSmartPointer<vtkSTLReader>::New();
  std::string in (input_name_1 + ".stl");
  sr_1->SetFileName(input_name_1.c_str());
  sr_1->Update();

  // convert unstructured grid to polydata
  vtkSmartPointer<vtkDataSetSurfaceFilter> sf =
    vtkSmartPointer<vtkDataSetSurfaceFilter>::New();
  sf->SetInputData(sr_1->GetOutput());
  sf->Update();


  ///////

  double ox = atof (argv[2]);
  double oy = atof (argv[3]);
  double oz = atof (argv[4]);

  double dy = atof (argv[5]);//0.01;
  double dz = atof (argv[6]);
  int n = atoi(argv[7]);
  int m = atoi(argv[8]);

  vtkSmartPointer<vtkAppendPolyData> af =
        vtkSmartPointer<vtkAppendPolyData>::New();

  for (int i=0; i<n; i++){
    for (int j=0; j<m; j++){

      vtkSmartPointer<vtkTransform> translation =
        vtkSmartPointer<vtkTransform>::New();
      translation->Translate(0 + ox - 0.3*j*dz , i*dy + oy, j*dz + oz );
```

```

      vtkSmartPointer<vtkTransformPolyDataFilter> transformFilter =
        vtkSmartPointer<vtkTransformPolyDataFilter>::New();
      transformFilter->SetInputConnection( sf->GetOutputPort() );
      transformFilter->SetTransform( translation );
      transformFilter->Update();

      af->AddInputData(transformFilter->GetOutput());
      af->Update();
    }
  }

   vtkSmartPointer<vtkSTLWriter> sw =
        vtkSmartPointer<vtkSTLWriter>::New();
   sw->SetFileName( "RODS.stl" );
   sw->SetInputData( af->GetOutput() );
   sw->SetFileTypeToBinary();
   sw->Write();

  return EXIT_SUCCESS;
}

void Other();
void Sphere();
void Cone();
void Ellipsoid();
void Cylinder();
void HyperboloidOneSheet();
void HyperboloidTwoSheets();
void HyperbolicParaboloid();
void EllipticParaboloid();

void PlotFunction(vtkQuadric* quadric, double value);


int main (int, char *[])
{

  Cylinder();

  return 0;
}

void Cylinder()
{
  // create the quadric function definition
```

```
  vtkSmartPointer<vtkQuadric> quadric = vtkSmartPointer<vtkQuadric>::New();
  quadric->SetCoefficients(1,1,0,0,0,0,0,0,0,0);

  // F(x,y,z) = a0*x^2 + a1*y^2 + a2*z^2 + a3*x*y + a4*y*z + a5*x*z + a6*x + a7*y + a8*z + a9
  // F(x,y,z) = 1*x^2 + 1*y^2

  PlotFunction(quadric, 1);
}

void PlotFunction(vtkQuadric* quadric, double value)
{

  // sample the quadric function
  vtkSmartPointer<vtkSampleFunction> sample = vtkSmartPointer<vtkSampleFunction>::New();
  sample->SetSampleDimensions(25,25,1000);
  sample->SetImplicitFunction(quadric);
  //double xmin = 0, xmax=1, ymin=0, ymax=1, zmin=0, zmax=1;
  double xmin = -1, xmax=1, ymin=-1, ymax=1, zmin=0, zmax=200;
  //double xmin = -10, xmax=10, ymin=-10, ymax=10, zmin=-10, zmax=10;
  sample->SetModelBounds(xmin, xmax, ymin, ymax, zmin, zmax);

  // Create five surfaces F(x,y,z) = constant between range specified
  /*
  vtkContourFilter *contours = vtkContourFilter::New();
  contours->SetInput(sample->GetOutput());
  contours->GenerateValues(5, 0.0, 1.2);
  */

  //create the 0 isosurface
  vtkSmartPointer<vtkContourFilter> contours = vtkSmartPointer<vtkContourFilter>::New();
  contours->SetInputConnection(sample->GetOutputPort());
  contours->GenerateValues(1, value, value);

  // write the detected boundary edges
  vtkSmartPointer<vtkSTLWriter> writer = vtkSmartPointer<vtkSTLWriter>::New();
  writer->SetInputConnection(contours->GetOutputPort());
  writer->SetFileName("kin.stl");
  writer->Write();

}
```

```
int main (int argc, char *argv[])
{
  std::cout << "argc = " << argc << std::endl;

  // PolyData to process
  std::string input_name1(argv[1]);
  std::cout << "Reading stl file : " << input_name1 << std::endl;
  vtkSmartPointer<vtkSTLReader> stlReader1 =
    vtkSmartPointer<vtkSTLReader>::New();
  stlReader1->SetFileName(input_name1.c_str());
  stlReader1->Update();
  vtkSmartPointer<vtkPolyData> pd1;
  pd1 = stlReader1->GetOutput();

  // Implicit function that will be used to slice the mesh
  vtkSmartPointer<vtkImplicitPolyDataDistance> implicitPolyDataDistance =
    vtkSmartPointer<vtkImplicitPolyDataDistance>::New();
  implicitPolyDataDistance->SetInput(pd1);

  // generate random points inside a box around the vestibule and valve
  // random points should be inside a cube of center (0.00 0.008 0.015)
  // and cube has length x=0.02 y=0.03 z=0.03
  srand(time(NULL)); // initialize random seed
  double lX = 0.02;
  double lY = 0.03;
  double lZ = 0.03;
  double centX = 0.0;
  double centY = 0.008;
  double centZ = 0.015;

  double sX = centX - lX/2;
  double sY = centY - lY/2;
  double sZ = centZ - lZ/2;

  std::ofstream pointsFile;
  pointsFile.open("injectionPoistions.txt");
  int count = 0;
  int evaluation = 0;

  while (count < 200 ) {

        std::cout << "evaluating " << evaluation <<  " .. " << std::endl;
```

```

        double randX = ((double) rand() / (RAND_MAX)); // random number between zero and one
        double randY = ((double) rand() / (RAND_MAX));
        double randZ = ((double) rand() / (RAND_MAX));

        double x = sX + randX*lX;
        double y = sY + randY*lY;
        double z = sZ + randZ*lZ;

        std::cout << x << " " << y << " " << z << std::endl;
        double p[3];
        p[0] = x;
        p[1] = y;
        p[2] = z;

        double signedDistance = implicitPolyDataDistance->EvaluateFunction(p);

        if ( signedDistance < -0.001 ){
                // add this point to the point list
                std::cout << "this point is inside! " << std::endl;
                pointsFile << x << " " << y << " " << z << std::endl;
                count ++;
        }

        evaluation ++;

  }
  // **********************************************************
  return EXIT_SUCCESS;
}

int main ( int argc, char *argv[] )
{
  std::cout << "grid std: usuage: ./obj input ox oy oz dy dz ny nz" << std::endl;

  std::string input_name_1 = argv[1];

  vtkSmartPointer<vtkSTLReader> sr_1 =
    vtkSmartPointer<vtkSTLReader>::New();
  std::string in (input_name_1 + ".stl");
  sr_1->SetFileName(input_name_1.c_str());
  sr_1->Update();

```

```
  // convert unstructured grid to polydata
  vtkSmartPointer<vtkDataSetSurfaceFilter> sf =
    vtkSmartPointer<vtkDataSetSurfaceFilter>::New();
  sf->SetInputData(sr_1->GetOutput());
  sf->Update();


  ///////

  double scale = atof (argv[2]);

  vtkSmartPointer<vtkTransform> translation =
        vtkSmartPointer<vtkTransform>::New();
  translation->Scale(1 , scale, scale );

  vtkSmartPointer<vtkTransformPolyDataFilter> transformFilter =
        vtkSmartPointer<vtkTransformPolyDataFilter>::New();
  transformFilter->SetInputConnection( sf->GetOutputPort() );
  transformFilter->SetTransform( translation );
  transformFilter->Update();

   vtkSmartPointer<vtkSTLWriter> sw =
        vtkSmartPointer<vtkSTLWriter>::New();
   sw->SetFileName( argv[3] );
   sw->SetInputData( transformFilter->GetOutput() );
   sw->SetFileTypeToBinary();
   sw->Write();

  return EXIT_SUCCESS;
}

int main (int argc, char *argv[])
{
  // PolyData to process
  std::string inputName1(argv[1]);
  std::cout << "Reading stl file : " << inputName1 << std::endl;
  vtkSmartPointer<vtkSTLReader> stlReader1 =
    vtkSmartPointer<vtkSTLReader>::New();
  stlReader1->SetFileName(inputName1.c_str());
  stlReader1->Update();
  vtkSmartPointer<vtkPolyData> polyData1;
  polyData1 = stlReader1->GetOutput();

  // PolyData to process
  std::string inputName2(argv[2]);
  std::cout << "Reading stl file : " << inputName2 <<
```

```
std::endl;
  vtkSmartPointer<vtkSTLReader> stlReader2 =
    vtkSmartPointer<vtkSTLReader>::New();
  stlReader2->SetFileName(inputName2.c_str());
  stlReader2->Update();
  vtkSmartPointer<vtkPolyData> polyData2;
  polyData2 = stlReader2->GetOutput();

// ********
        vtkSmartPointer<vtkFeatureEdges> boundaryEdges1 =
                vtkSmartPointer<vtkFeatureEdges>::New();
        boundaryEdges1->SetInputData(polyData1);
        boundaryEdges1->BoundaryEdgesOn();
        boundaryEdges1->FeatureEdgesOff();
        boundaryEdges1->NonManifoldEdgesOff();
        boundaryEdges1->ColoringOff();
        boundaryEdges1->Update();

        vtkSmartPointer<vtkFeatureEdges> boundaryEdges2 =
                vtkSmartPointer<vtkFeatureEdges>::New();
        boundaryEdges2->SetInputData(polyData2);
        boundaryEdges2->BoundaryEdgesOn();
        boundaryEdges2->FeatureEdgesOff();
        boundaryEdges2->NonManifoldEdgesOff();
        boundaryEdges2->ColoringOff();
        boundaryEdges2->Update();

// ********

        vtkSmartPointer<vtkPolyDataConnectivityFilter> connectivityFilter1 =
vtkSmartPointer<vtkPolyDataConnectivityFilter>::New();
        connectivityFilter1->SetInputData(boundaryEdges1->GetOutput());
        connectivityFilter1->SetExtractionModeToSpecifiedRegions();
        connectivityFilter1->AddSpecifiedRegion(0);
        connectivityFilter1->Update();



        vtkSmartPointer<vtkPolyDataConnectivityFilter> connectivityFilter2 =
vtkSmartPointer<vtkPolyDataConnectivityFilter>::New();
        connectivityFilter2->SetInputData(boundaryEdges2->GetOutput());
```

```
        connectivityFilter2->SetExtractionModeToSpecifiedRegions();
        connectivityFilter2->AddSpecifiedRegion(0);
        connectivityFilter2->Update();

// *********

        vtkSmartPointer<vtkCleanPolyData> cleanPolyData1 =
                vtkSmartPointer<vtkCleanPolyData>::New();
        cleanPolyData1->SetInputData(connectivityFilter1->GetOutput());
        cleanPolyData1->Update();

            // Write the file
  vtkSmartPointer<vtkXMLPolyDataWriter> writer1 =
    vtkSmartPointer<vtkXMLPolyDataWriter>::New();
  writer1->SetFileName("test1.vtp");
  writer1->SetInputData(cleanPolyData1->GetOutput());
  // Optional - set the mode. The default is binary.
  //writer->SetDataModeToBinary();
  //writer->SetDataModeToAscii();
  writer1->Write();

        vtkSmartPointer<vtkCleanPolyData> cleanPolyData2 =
                vtkSmartPointer<vtkCleanPolyData>::New();
        cleanPolyData2->SetInputData(connectivityFilter2->GetOutput());
        cleanPolyData2->Update();

            // Write the file
  vtkSmartPointer<vtkXMLPolyDataWriter> writer2 =
    vtkSmartPointer<vtkXMLPolyDataWriter>::New();
  writer2->SetFileName("test2.vtp");
  writer2->SetInputData(cleanPolyData2->GetOutput());
  // Optional - set the mode. The default is binary.
  //writer->SetDataModeToBinary();
  //writer->SetDataModeToAscii();
  writer2->Write();

// ********

        vtkSmartPointer<vtkAppendPolyData> appendPolyDataFilter =
            vtkSmartPointer<vtkAppendPolyData>::New();
        appendPolyDataFilter->AddInputData( cleanPolyData1->GetOutput() );
        appendPolyDataFilter->AddInputData( cleanPolyData2->GetOutput() );
```

```
        appendPolyDataFilter->Update();

        // ********

        vtkSmartPointer<vtkDelaunay2D> delauny =
            vtkSmartPointer<vtkDelaunay2D>::New();
        delauny->SetInputData(appendPolyDataFilter->GetOutput());
        delauny->SetProjectionPlaneMode(VTK_BEST_FITTING_PLANE);
        delauny->Update();

/*
        std::ostringstream ss;
        std::string out (argv[1]);
        ss << out << ".stl";
        out  = ss.str();

        std::string name1(out);
        vtkSmartPointer<vtkSTLWriter> writer =
                vtkSmartPointer<vtkSTLWriter>::New();
        writer1->SetFileName(name1.c_str());
        std::cout << "writing .. " << std::endl;
        writer->SetInputData(delauny->GetOutput());
        writer->Write();

*/
  return EXIT_SUCCESS;
}

int main(int argc, char * argv[])
{
    std::cout << "reversing normals .." << std::endl;
    // PolyData to process
    std::string input_name1(argv[1]);
    std::cout << "reading stl file : " << input_name1 << std::endl;
    vtkSmartPointer<vtkSTLReader> stlReader1 =
    vtkSmartPointer<vtkSTLReader>::New();
    stlReader1->SetFileName(input_name1.c_str());
    stlReader1->Update();
    vtkSmartPointer<vtkPolyData> pd1;
    pd1 = stlReader1->GetOutput();


  vtkSmartPointer<vtkReverseSense> reverseSense =
    vtkSmartPointer<vtkReverseSense>::New();
  reverseSense->SetInputData(pd1);
```

```
  reverseSense->ReverseNormalsOn();
  reverseSense->Update();

    /////
    std::string outname(argv[2]);
    std::string outSTL = outname;
    vtkSmartPointer<vtkSTLWriter> sw2 = vtkSmartPointer<vtkSTLWriter>::New();
    sw2->SetFileName(outSTL.c_str());
    std::cout << "writing stl .. " << std::endl;
    sw2->SetInputData(reverseSense->GetOutput());
    sw2->Write();

  return EXIT_SUCCESS;
}

/**
** end of VTK codes
**/

```

Appendix C: Scripts (BASH)

```
#!/bin/bash

##############################################################
# main script which starts the optimization iteration
# this script is called by dakota as part of evaluation process
# the "|| true" to ensure iteration will not stop for minor errors
##############################################################

set -e
set -o errexit

echo "----------------------------------------"
echo "MAIN-dak ..."
echo "----------------------------------------"

START_LINE_NUM=1;
N_LINES=1;
N_LINES_OBS=14;
N_LINES_RODS=1;
MODE=1;

# addresses are absolute to ensure the correctness in alpha phase
# they should be changed to relative for robustness

HOME_DIR="/media/milad/ssd0/master_folder/Optimization"
FLUID_DIR="$HOME_DIR/Fluid"
PARTICLE_DIR="$HOME_DIR/Particle"

# method from dakota for text parsing
#creating init for main geom
dprepro $1 INITIATE.template INITIATE

# creating init for obstacle 0
dprepro $1 INITIATE_OBS.template INITIATE_OBS

#creating init for rods
dprepro $1 INITIATE_RODS.template INITIATE_RODS

cp $HOME_DIR/INITIATE $HOME_DIR/pts.lis
cp pts.lis pts.lis.$2

cp $HOME_DIR/INITIATE_OBS $HOME_DIR/pts_obs.lis
cp pts_obs.lis pts_obs.lis.$2

cp $HOME_DIR/INITIATE_RODS $HOME_DIR/pts_rods.lis
```

```
cp pts_rods.lis pts_rods.lis.$2

touch ${HOME_DIR}/norm.tmp
touch ${HOME_DIR}/results.out

$FLUID_DIR/CREATE_NEW_FLOW_CASE.sh

# loop over variables
for L in `seq 1 $N_LINES`
do
        LINE=`(cat pts.lis | head -$L | tail -1)`
        P_FLAG=`echo $LINE |cut -d " " -f1`
        NEW_X=`echo $LINE |cut -d " " -f2`
        NEW_Y=`echo $LINE |cut -d " " -f3`
        NEW_Z=`echo $LINE |cut -d " " -f4`
        $FLUID_DIR/CHANGE_A_POINT.sh $P_FLAG $NEW_X blockMeshDict_I
done

#rm obs_points.txt || true
for L in `seq 1 $N_LINES_OBS`
do
        LINE=`(cat pts_obs.lis | head -$L | tail -1)`
        P_FLAG=`echo $LINE |cut -d " " -f1`
        NEW_X=`echo $LINE |cut -d " " -f2`
        $FLUID_DIR/CHANGE_A_POINT.sh $P_FLAG $NEW_X blockMeshDict_OBS
        #echo "$NEW_X" >> obs_points.txt
done

rm clipPlane.txt || true

$HOME_DIR/obs_manager "gen_clip_plane"

for L in `seq 1 $N_LINES_RODS`
do
        LINE=`(cat pts_rods.lis | head -$L | tail -1)`
        FLAG=`echo $LINE |cut -d " " -f1`
        NEW=`echo $LINE |cut -d " " -f2`
        $FLUID_DIR/TMP_CASE/constant/triSurface
                > /CHANGE_STH.sh $FLAG $NEW makeRods.sh
done

cp clipPlane.txt $HOME_DIR/CASE/constant/triSurface/

# | tee $HOME_DIR/flog_$PTS_LIS_I
$FLUID_DIR/PERFORM_FLOW_CASE.sh $MODE
```

```
#save a copy of flow case
cp -r CASE CASE_$2 || true

# this removes residuals from previous step if any
rm -rf plog* PCASE* || true

$PARTICLE_DIR/PARTICLE_MAIN.sh

mv CASE_$2 "/media/milad/Seagate Backup Plus Drive/OPT_CASES" || true

echo "Adding norm ... "

# postproc
$HOME_DIR/opt_manager "append_norm" $2

# preparing result of this iteration for postprocessing code

mv vesti.txt vesti.$2 || true
mv valve.txt valve.$2 || true
mv olf.txt olf.$2 || true
mv turbinates.txt turbinates.$2 || true
mv naso.txt naso.$2 || true
mv outlet.txt outlet.$2 || true

mv vestiEr.txt vestiEr.$2 || true
mv valveEr.txt valveEr.$2 || true
mv olfEr.txt olfEr.$2 || true
mv turbinatesEr.txt turbinatesEr.$2 || true
mv nasoEr.txt nasoEr.$2 || true
mv outletEr.txt outletEr.$2 || true

mv vestivel0.txt vestivel0.$2 || true
mv valvevel0.txt valvevel0.$2 || true
mv olfvel0.txt olfvel0.$2 || true
mv turbinatesvel0.txt turbinatesvel0.$2 || true
mv nasovel0.txt nasovel0.$2 || true
mv outletvel0.txt outletvel0.$2 || true

mv vestivel2.txt vestivel2.$2 || true
mv valvevel2.txt valvevel2.$2 || true
mv olfvel2.txt olfvel2.$2 || true
mv turbinatesvel2.txt turbinatesvel2.$2 || true
mv nasovel2.txt nasovel2.$2 || true
mv outletvel2.txt outletvel2.$2 || true

mv vestivel3.txt vestivel3.$2 || true
mv valvevel3.txt valvevel3.$2 || true
```

```
mv olfvel3.txt olfvel3.$2 || true
mv turbinatesvel3.txt turbinatesvel3.$2 || true
mv nasovel3.txt nasovel3.$2 || true
mv outletvel3.txt outletvel3.$2 || true


rm -rf CASE

rm norm.tmp
```

```bash
#!/bin/bash

###########################################################
# perform stl manipulation and CFD cases
###########################################################
# FILE: PERFORM_FLOW_CASE.sh
# Bash script for creating new case from template, go through vtk, run flow case
# blockmesh to extract surface

set -e
set -o errexit

END_T=80;
DIREC="/media/milad/ssd0/master_folder/Optimization/CASE"
P_HOME="/media/milad/ssd0/master_folder/Optimization"
TRI="/media/milad/ssd0/master_folder/Optimization/CASE/constant/triSurface"

MODE=$1

if [ $MODE -lt 0 ]
then
        cp $P_HOME/ready.stl $TRI/smooth.stl
fi

if [ $MODE -gt 0 ]
then
        mv $DIREC/0 $DIREC/0.org

        echo "creating obstacle "
        cp $DIREC/system/blockMeshDict_OBS $DIREC/system/blockMeshDict
        blockMesh -case $DIREC
        foamToVTK -case $DIREC
        cp $DIREC/VTK/OBS/OBS_0.vtk $DIREC/constant/triSurface/

        echo "Changing blockMesh dicttionary file for main branch .."
        rm $DIREC/system/blockMeshDict
        cp $DIREC/system/blockMeshDict_I $DIREC/system/blockMeshDict
        blockMesh -case $DIREC
        echo "Running foamToVTK"
        foamToVTK -case $DIREC

        # convert vtk files to stl files
```

```
        echo "Copying vtk files into triSurface"
        cp $DIREC/VTK/VESTIBULE/VESTIBULE_0.vtk $DIREC/constant/triSurface/
        cp $DIREC/VTK/VALVE/VALVE_0.vtk $DIREC/constant/triSurface/
        cp $DIREC/VTK/ANTERIOR/ANTERIOR_0.vtk $DIREC/constant/triSurface/
        cp $DIREC/VTK/POSTERIOR/POSTERIOR_0.vtk $DIREC/constant/triSurface/
        cp $DIREC/VTK/OLF/OLF_0.vtk $DIREC/constant/triSurface/
        cp $DIREC/VTK/NASO/NASO_0.vtk $DIREC/constant/triSurface/
        cp $DIREC/VTK/INLET/INLET_0.vtk $DIREC/constant/triSurface/
        cp $DIREC/VTK/OUTLET/OUTLET_0.vtk $DIREC/constant/triSurface/

        echo "appending, smoothing and clipping .."
        cd $TRI

        ITER=10000;

        ./makeSTLs.sh .    ## makes stl files from vtk files generated by blockMesh

        ./append_IO INLET.stl VESTIBULE.stl t1.stl
        ./append_IO t1.stl VALVE.stl t2.stl
        ./append_IO t2.stl ANTERIOR.stl t3.stl
        ./append_IO t3.stl OLF.stl t4.stl
        ./append_IO t4.stl POSTERIOR.stl t5.stl
        ./append_IO t5.stl NASO.stl t6.stl
        ./append_IO t6.stl OUTLET.stl all.stl

        cp all.stl all-bkp.stl

        ./smoothAll all.stl all.stl 2000 0.01

        ./smoothAll OBS.stl OBS.stl 2000 0.01
        cp OBS.stl OBS-bkp.stl

        cp all.stl smooth-ini.stl

        ./punchClose.sh all.stl OBS.stl

        ./makeRods.sh

         cp RODS.stl RODS-bkp.stl
```

```
        ./CapClip RODS.stl "clipPlane.txt" # clip rods to stay within the obs

        ./fillHoles RODS.stl RODS.stl

#       rm clipPlane.txt

        mv all.stl smooth.stl
fi

cd $TRI
./clips #this defines the patch stls finally
./append_IO POSTERIOR.stl ANTERIOR.stl TURBINATES.stl

#./scaleSTL VESTIBULE.stl 0.9 VESTIBULE.stl
#./scaleSTL VALVE.stl 0.9 VALVE.stl
#./scaleSTL OLF.stl 0.9 OLF.stl
#./scaleSTL TURBINATES.stl 0.9 TURBINATES.stl
#./scaleSTL NASO.stl 0.9 NASO.stl
#./scaleSTL OUTLET.stl 0.9 OUTLET.stl
#./scaleSTL INLET.stl 0.9 INLET.stl
#./scaleSTL OBS.stl 0.9 OBS.stl
#./scaleSTL RODS.stl 0.9 RODS.stl

cd $P_HOME

echo "Removing vtk files from triSurface"
#rm $TRI/*vtk

echo "Changing blockMesh dicttionary file for snappyHexMesh.."
mv $DIREC/system/blockMeshDict $DIREC/system/blockMeshDict_surf
mv $DIREC/system/blockMeshDict_snappy $DIREC/system/blockMeshDict

# this blockmesh is for snappyhexmesh boundaries
echo "Removing VTK directory"
rm -rf $DIREC/VTK

#echo "Renaming 0 to 0.org"
#mv $DIREC/0 $DIREC/0.org

echo "Running blockMesh"
blockMesh -case $DIREC

echo "Running surfaceFeatureExtract"
```

```
surfaceFeatureExtract -case $DIREC

echo "decomposing case for meshing"
decomposePar -case $DIREC

echo "Running snappyHexMesh"
foamJob -case $DIREC -p -s snappyHexMesh
reconstructParMesh -case $DIREC
rm -rf $DIREC/proc*

echo "Removing previous polyMesh data "
rm -rf $DIREC/constant/polyMesh/*

echo "Copying snappyHexMesh data to polyMesh directory"
cp $DIREC/2/polyMesh/* $DIREC/constant/polyMesh/

echo "Removing 1 and 2 directories"
rm -rf $DIREC/1 $DIREC/2

echo "Renaming 0.org to 0 "
mv $DIREC/0.org $DIREC/0

echo "Running flow case: simpleFoam .."
decomposePar -case $DIREC
foamJob -case $DIREC -p -s simpleFoam

# foamMonitor -l postProcessing/residuals/0/residuals.dat

reconstructPar -case $DIREC -latestTime

rm -rf $DIREC/proc*
rm -rf $DIREC/0 $DIREC/$END_T/uniform
mv $DIREC/$END_T $DIREC/0

echo "foamToVTK"
foamToVTK -case $DIREC -latestTime

mv $DIREC/system/controlDict $DIREC/system/controlDict_fluid
mv $DIREC/system/controlDict_particles $DIREC/system/
controlDict
```

```bash
#!/bin/bash

##################################################################
# main script for handeling particle tracking in parallel
##################################################################

# File: PARTICLE_MAIN.sh
# $1 flow case number which this particle tracking is performed on
# this script utilize the idle threads

set -e
set -o errexit

unset NPROC N_RUN_PER_PROC START_
unset LINE_NUM MAXRUN HOME_DIR PARTICLE_
unset DIR LINE LINE_NUM
unset SIZE VEL POSITION_LABEL
unset POSITIONX POSITIONY POSITIONZ DIREC
unset U G Y pid waitForIdleProc foundIdle

echo "Performing particle tracking cases .."

N_RUN_PER_PROC=1;
START_LINE_NUM=10; # line start of the parameters
NPROC=15; # number of processors to be involved
N1=$NPROC;
NN1=$(($N1));
MAXRUNS=4000; # maximum number of cases

HOME_DIR="/media/milad/ssd0/master_folder/Optimization"
PARTICLE_DIR="$HOME_DIR/Particle"

#initializing process ids
for G in `seq 1 $NPROC`
do
        pid[$G]=0;
done

function waitForIdleProc {
        echo "searching ..."
        foundIdle=0;
        while [ $foundIdle -eq 0 ]
        do
                for Y in `seq 1 ${NN1}`
                do
                        if [ ${pid[$Y]} -eq 0 ] || ! ps -p ${pid[$Y]}
```

```
                                > /dev/null; then
                                foundIdle=1;
                                icoUncoupledKinematicParcelFoam -case $1
                                > $2/plog_$3 2>&1 &
                                pid[$Y]=$!
                                echo "found idle!" && hostname
                                && echo "job pi $Y bg $! id $$"
                                break
                        fi
                done

                [ $foundIdle == 1 ] && break
                echo "waiting ..." && sleep 60;
        done
}

for U in `seq 1 $MAXRUNS`
do
        # get rid of large stuff which are hanging around for too long (15 min)
        find $HOME_DIR -type d -name "*PCASE*"
                > -mmin +15 -exec rm -rf {} +

        LINE_NUM=$(( $U - 1 + $START_LINE_NUM))
        LINE=`(cat $PARTICLE_DIR/particleParameters.lis
                | head -$LINE_NUM | tail -1)`

        SIZE=`echo $LINE |cut -d " " -f1`
        VEL=`echo $LINE |cut -d " " -f2`
        POSITION_LABEL=`echo $LINE |cut -d " " -f3`
        POSITIONX=`echo $LINE |cut -d " " -f4`
        POSITIONY=`echo $LINE |cut -d " " -f5`
        POSITIONZ=`echo $LINE |cut -d " " -f6`
        T="${POSITION_LABEL}_${SIZE}_${VEL}"
        DIREC="/media/milad/ssd0/master_folder/Optimization/PCASE_$T"
        cp -r "$HOME_DIR/CASE" $DIREC

        $PARTICLE_DIR/PARTICLE_SIZE_SET.sh $SIZE $DIREC
        $PARTICLE_DIR/PARTICLE_U0_SET.sh $VEL $DIREC
        $PARTICLE_DIR/PARTICLE_POSITION_SET.sh $POSITIONX $POSITIONY $POSITIONZ $DIREC
        echo "d $SIZE u0 $VEL posi $POSITION_LABEL"
        echo "case $U of $MAXRUNS ready"
        waitForIdleProc $DIREC $HOME_DIR $T $NPROC
```

```
        echo "- - - - - - - - - - -"
done

echo "waiting for all background jobs to finish ..." && wait
echo "finished!" && echo "- - - - - - - - - -"
```

```
#######################
# start of dakota input
#######################
# manually generated
# # sign shows is comment line
# # endofline character should be assigned as "\"
environment \
        tabular_graphics_data \
\
method \
        #conmin_mfd \
        optpp_q_newton \
                max_iterations = 3000 \
                convergence_tolerance = 1e-7 \
                search_method   value_based_line_search \
                merit_function  argaez_tapia \
model \
        single \
\
variables, \
        continuous_design = 15 \
                \
                initial_point   \
\
#                               ( 0.0035 0.028 0.026 )  // 0 --> 17
#                               ( 0.005 0.044 0.026 )  // 1 --> 80
                                0.002 0.044 0.026 \
                                0.002 0.028 0.026 \
\
#                               ( -0.0025 0.028 0.046 )  // 4
#                               ( 0.000 0.044 0.046 )  // 5
                                -0.001 0.044 0.046 \
                                -0.004 0.028 0.048 \
                                -0.003 \
                                -0.001 \
                                1 \
\
                upper_bounds    \
\
#                               0.009 0.034 0.032 \
#                               0.012 0.047 0.032 \
                                0.005 0.046 0.032 \
                                0.006 0.034 0.032 \
\
#                               0.01 0.034 0.055 \
#                               0.01 0.044 0.055 \
```

```
                                0.005 0.044 0.055 \
                                0.005 0.034 0.055 \
                                0.000 \
                                -0.001 \
                                5 \
                lower_bounds    \
\
#                               0.001 0.028 0.023 \
#                               0.004 0.040 0.023 \
                                -0.005 0.040 0.023 \
                                -0.004 0.028 0.023 \
\
#                               -0.004 0.028 0.040 \
#                               -0.004 0.038 0.040 \
                                -0.006  0.038 0.040 \
                                -0.006 0.028 0.040 \
                                -0.007 \
                                -0.007 \
                                0.5 \
\
                descriptors     \
\
#                               'obi0' 'obj0' 'obk0' \
#                               'obi1' 'obj1' 'obk1' \
                                'obi2' 'obj2' 'obk2' \
                                'obi3' 'obj3' 'obk3' \
\
#                               'obi4' 'obj4' 'obk4' \
#                               'obi5' 'obj5' 'obk5' \
                                'obi6' 'obj6' 'obk6' \
                                'obi7' 'obj7' 'obk7' \
                                'obspi0307'
                                'obspi0602'
                                'rscaleFactor'
\
                #linear_inequality_constraint_matrix = \
                # x constraints for the valid wall geometry Vertices
# 1  0  0  0    0  0  0  0    -1  0  0  0    0  0  0  0   0 0 0   0 0 0   0 0 0  0 0 0    0 0 0   0 0 0   0 0 0  0 0 0    0\
# 0  0  0  0    1  0  0  0    0  0  0  0    -1  0  0  0   0 0 0   0 0 0   0 0 0  0 0 0    0 0 0   0 0 0   0 0 0  0 0 0    0\
# 0  1  0  0    0  0  0  0    0  -1  0  0    0  0  0  0   0 0 0   0 0 0   0 0 0  0 0 0    0 0 0   0 0 0   0 0 0  0 0 0    0\
# 0  0  0  0    0  1  0  0    0  0  0  0    0  -1  0  0
```

```
0 0 0   0 0 0   0 0 0  0 0 0     0 0 0   0 0 0   0 0 0  0 0 0   0\
\
                            # x constraints for wall geometry Splines
                            # constraints for obs wihtin its geometry
                            # x
                            #1 0 0   0 0 0   0 0 0  -1 0 0     0 0 0   0 0 0   0 0 0  0 0 0   0\
                            #0 0 0   1 0 0   -1 0 0  0 0 0     0 0 0   0 0 0   0 0 0  0 0 0   0\
                            #0 0 0   0 0 0   0 0 0  0 0 0     1 0 0   0 0 0   0 0 0  -1 0 0   0\
                            #0 0 0   0 0 0   0 0 0  0 0 0     0 0 0   1 0 0   -1 0 0  0 0 0   0\
                            # y
                            #0 -1 0   0 1 0   0 0 0  0 0 0     0 0 0   0 0 0   0 0 0  0 0 0   0\
                            #0 0 0   0 0 0   0 1 0  0 -1 0     0 0 0   0 0 0   0 0 0  0 0 0   0\
                            #0 0 0   0 0 0   0 0 0  0 0 0     0 -1 0   0 1 0   0 0 0  0 0 0   0\
                            #0 0 0   0 0 0   0 0 0  0 0 0     0 0 0   0 0 0   0 1 0  0 -1 0   0\
                            # z
                            #0 0 -1   0 0 0   0 0 0  0 0 0     0 0 1   0 0 0   0 0 0  0 0 0   0\
                            #0 0 0   0 0 -1   0 0 0  0 0 0     0 0 0   0 0 1   0 0 0  0 0 0   0\
                            #0 0 0   0 0 0   0 0 -1  0 0 0     0 0 0   0 0 0   0 0 1  0 0 0   0\
                            #0 0 0   0 0 0   0 0 0  0 0 -1     0 0 0   0 0 0   0 0 0  0 0 1   0\
\
\
                #linear_inequality_upper_bounds =    0.03 0.03 0.03 0.03   0.03 0.03 0.03 0.03   0.03 0.03 0.03 0.03 \
                #linear_inequality_lower_bounds =    0.001 0.001 0.001 0.001   0.001 0.001 0.001 0.001   0.001 0.001 0.001 0.001 \
\
interface, \
        fork \
                analysis_drivers = 'MAIN-dak.sh' \
                parameters_file  = 'params.in' \
                results_file     = 'results.out' \
                file_tag \
```

```
                file_save \
\
responses       \
        objective_functions = 1 \
        descriptors = 'norm' \
        numerical_gradients \
                method_source dakota \
                interval_type forward \
                fd_gradient_step_size = 5e-2 # was 1e-4 \
#       no_gradient
        no_hessians
#
#######################
# end of dakota input
#######################
```

Appendix D: Postprocessing (BASH, Python)

```
#!/bin/bash

########################################################
# automated plot generator
# milad kiaee darunkola kiaeedar@ualberta.ca
########################################################

rm *png
rm Snap*

cp ../ref_vesti.txt refvesti.dlt
cp ../ref_valve.txt refvalve.dlt
cp ../ref_olf.txt refolf.dlt
cp ../ref_turbinates.txt refturbinates.dlt
cp ../ref_naso.txt refnaso.dlt
cp ../ref_outlet.txt refoutlet.dlt

./runPlot.sh refvesti.dlt matplot.plt
mv tmp.png refvesti.png
convert refvesti.png -resize 200x200 refvesti.png

./runPlot.sh refvalve.dlt matplot.plt
mv tmp.png refvalve.png
convert refvalve.png -resize 200x200 refvalve.png

./runPlot.sh refolf.dlt matplot.plt
mv tmp.png refolf.png
convert refolf.png -resize 200x200 refolf.png

./runPlot.sh refturbinates.dlt matplot.plt
mv tmp.png refturbinates.png
convert refturbinates.png -resize 200x200 refturbinates.pn

./runPlot.sh refnaso.dlt matplot.plt
mv tmp.png refnaso.png
convert refnaso.png -resize 200x200 refnaso.png

./runPlot.sh refoutlet.dlt matplot.plt
mv tmp.png refoutlet.png
convert refoutlet.png -resize 200x200 refoutlet.png

for i in `seq 1 $1`
do
        python snap.py $i

        convert -size 1000x1800 'Snap_'$i'.png'

#       convert -size 1000x1800 white 'Snap_'$i'.png'
```

```
	line=`(cat ../results.out.$i | head -1 | tail -1)`
	norm=`echo $line |cut -d " " -f1`
	tmp=$norm
	norm=$(bc <<< "scale=2;$tmp*100")

	TEXT1="iteration $i , deviation = $norm %"

	convert -font helvetica -fill white -pointsize 20
	> -draw "text 10,30 '$TEXT1'/6.0"
	> 'Snap_'$i.png 'Snap_'$i.png

	./runPlot.sh ../vesti.results.out.$i matplot.plt
	mv tmp.png vesti$i.png
	convert vesti$i.png -resize 800x800 vesti$i.png

	./runPlot.sh ../valve.results.out.$i matplot.plt
	mv tmp.png valve$i.png
	convert valve$i.png -resize 800x800 valve$i.png

	./runPlot.sh ../olf.results.out.$i matplot.plt
	mv tmp.png olf$i.png
	convert olf$i.png -resize 800x800 olf$i.png

	./runPlot.sh ../turbinates.results.out.$i matplot.plt
	mv tmp.png turbinates$i.png
	convert turbinates$i.png -resize 800x800 turbinates$i.png

	./runPlot.sh ../naso.results.out.$i matplot.plt
	mv tmp.png naso$i.png
	convert naso$i.png -resize 800x800 naso$i.png

	./runPlot.sh ../outlet.results.out.$i matplot.plt
	mv tmp.png outlet$i.png
	convert outlet$i.png -resize 800x800 outlet$i.png

	### Eror plot ###
	#./runPlot.sh ../vestiEr.results.out.$i errorplot.plt
	#mv tmp.png vestiEr$i.png
	#convert vestiEr$i.png -resize 300x300 vestiEr$i.png

	#./runPlot.sh ../valveEr.results.out.$i errorplot.plt
	#mv tmp.png valveEr$i.png
	#convert valveEr$i.png -resize 300x300 valveEr$i.png

	#./runPlot.sh ../olfEr.results.out.$i errorplot.plt
	#mv tmp.png olfEr$i.png
```

```
        #convert olfEr$i.png -resize 300x300 olfEr$i.png

        #./runPlot.sh ../turbinatesEr.results.out.$i errorplot.plt
        #mv tmp.png turbinatesEr$i.png
        #convert turbinatesEr$i.png -resize 300x300 turbinatesEr$i.png

        #./runPlot.sh ../nasoEr.results.out.$i errorplot.plt
        #mv tmp.png nasoEr$i.png
        #convert nasoEr$i.png -resize 300x300 nasoEr$i.png

        #./runPlot.sh ../outletEr.results.out.$i errorplot.plt
        #mv tmp.png outletEr$i.png
        #convert outletEr$i.png -resize 300x300 outletEr$i.png

        ######
        ### scatter plot ###
        ./runPlot.sh ../vestivel0.results.out.$i scatterplot.plt
        mv tmp.png vestivel0$i.png
        convert vestivel0$i.png -resize 800x800 vestivel0$i.png

        ./runPlot.sh ../valvevel0.results.out.$i scatterplot.plt
        mv tmp.png valvevel0$i.png
        convert valvevel0$i.png -resize 800x800 valvevel0$i.png

        ./runPlot.sh ../olfvel0.results.out.$i scatterplot.plt
        mv tmp.png olfvel0$i.png
        convert olfvel0$i.png -resize 800x800 olfvel0$i.png

        ./runPlot.sh ../turbinatesvel0.results.out.$i scatterplot.plt
        mv tmp.png turbinatesvel0$i.png
        convert turbinatesvel0$i.png -resize 800x800 turbinatesvel0$i.png

        ./runPlot.sh ../nasovel0.results.out.$i scatterplot.plt
        mv tmp.png nasovel0$i.png
        convert nasovel0$i.png -resize 800x800 nasovel0$i.png

```

```
        ./runPlot.sh ../outletvel0.results.out.$i scatterplot.plt
        mv tmp.png outletvel0$i.png
        convert outletvel0$i.png -resize 800x800 outletvel0$i.png
        ######
        ./runPlot.sh ../vestivel2.results.out.$i scatterplot.plt
        mv tmp.png vestivel2$i.png
        convert vestivel2$i.png -resize 800x800 vestivel2$i.png

        ./runPlot.sh ../valvevel2.results.out.$i scatterplot.plt
        mv tmp.png valvevel2$i.png
        convert valvevel2$i.png -resize 800x800 valvevel2$i.png

        ./runPlot.sh ../olfvel2.results.out.$i scatterplot.plt
        mv tmp.png olfvel2$i.png
        convert olfvel2$i.png -resize 800x800 olfvel2$i.png

        ./runPlot.sh ../turbinatesvel2.results.out.$i scatterplot.plt
        mv tmp.png turbinatesvel2$i.png
        convert turbinatesvel2$i.png -resize 800x800 turbinatesvel2$i.png

        ./runPlot.sh ../nasovel2.results.out.$i scatterplot.plt
        mv tmp.png nasovel2$i.png
        convert nasovel2$i.png -resize 800x800 nasovel2$i.png

        ./runPlot.sh ../outletvel2.results.out.$i scatterplot.plt
        mv tmp.png outletvel2$i.png
        convert outletvel2$i.png -resize 800x800 outletvel2$i.png
        ######
        ./runPlot.sh ../vestivel3.results.out.$i scatterplot.plt
        mv tmp.png vestivel3$i.png
        convert vestivel3$i.png -resize 800x800 vestivel3$i.png

        ./runPlot.sh ../valvevel3.results.out.$i scatterplot.plt
```

```
        mv tmp.png valvevel3$i.png
        convert valvevel3$i.png -resize 800x800 valvevel3$i.png

        ./runPlot.sh ../olfvel3.results.out.$i scatterplot.plt
        mv tmp.png olfvel3$i.png
        convert olfvel3$i.png -resize 800x800 olfvel3$i.png

        ./runPlot.sh ../turbinatesvel3.results.out.$i scatterplot.plt
        mv tmp.png turbinatesvel3$i.png
        convert turbinatesvel3$i.png -resize 800x800 turbinatesvel3$i.png

        ./runPlot.sh ../nasovel3.results.out.$i scatterplot.plt
        mv tmp.png nasovel3$i.png
        convert nasovel3$i.png -resize 800x800 nasovel3$i.png

        ./runPlot.sh ../outletvel3.results.out.$i scatterplot.plt
        mv tmp.png outletvel3$i.png
        convert outletvel3$i.png -resize 800x800 outletvel3$i.png
        ######

        convert Snap_$i.png vesti$i.png -geometry
        > 150x150+0+50 -composite Snap_$i.png
        convert Snap_$i.png refvesti.png -geometry
        > 150x150+200+50 -composite Snap_$i.png
#       convert Snap_$i.png vestiEr$i.png -geometry 250x250+1100+0 -composite Snap_$i.png
        convert Snap_$i.png vestivel0$i.png -geometry
        > 250x250+1100+0 -composite Snap_$i.png
        convert Snap_$i.png vestivel2$i.png -geometry
        > 250x250+1400+0 -composite Snap_$i.png
        convert Snap_$i.png vestivel3$i.png -geometry
        > 250x250+1700+0 -composite Snap_$i.png

        convert Snap_$i.png valve$i.png -geometry
        > 150x150+0+200 -composite Snap_$i.png
        convert Snap_$i.png refvalve.png -geometry
        > 150x150+200+200 -composite Snap_$i.png
#       convert Snap_$i.png valveEr$i.png -geometry 250x250+1100+200 -composite Snap_$i.png
        convert Snap_$i.png valvevel0$i.png -geometry
        > 250x250+1100+200 -composite Snap_$i.png
```

```
        convert Snap_$i.png valvevel2$i.png -geometry
        > 250x250+1400+200 -composite Snap_$i.png
        convert Snap_$i.png valvevel3$i.png -geometry
        > 250x250+1700+200 -composite Snap_$i.png

        convert Snap_$i.png olf$i.png -geometry
        > 150x150+0+400 -composite Snap_$i.png
        convert Snap_$i.png refolf.png -geometry
        > 150x150+200+400 -composite Snap_$i.png
#       convert Snap_$i.png olfEr$i.png -geometry 250x250+1100+400 -composite Snap_$i.png
        convert Snap_$i.png olfvel0$i.png -geometry
        > 250x250+1100+400 -composite Snap_$i.png
        convert Snap_$i.png olfvel2$i.png -geometry
        > 250x250+1400+400 -composite Snap_$i.png
        convert Snap_$i.png olfvel3$i.png -geometry
        > 250x250+1700+400 -composite Snap_$i.png

        convert Snap_$i.png turbinates$i.png -geometry
        > 150x150+0+600 -composite Snap_$i.png
        convert Snap_$i.png refturbinates.png -geometry
        > 150x150+200+600 -composite Snap_$i.png
#       convert Snap_$i.png turbinatesEr$i.png -geometry 250x250+1100+600 -composite Snap_$i.png
        convert Snap_$i.png turbinatesvel0$i.png -geometry
        > 250x250+1100+600 -composite Snap_$i.png
        convert Snap_$i.png turbinatesvel2$i.png -geometry
        > 250x250+1400+600 -composite Snap_$i.png
        convert Snap_$i.png turbinatesvel3$i.png -geometry
        > 250x250+1700+600 -composite Snap_$i.png

        convert Snap_$i.png naso$i.png -geometry
        > 150x150+0+800 -composite Snap_$i.png
        convert Snap_$i.png refnaso.png -geometry
        > 150x150+200+800 -composite Snap_$i.png
#       convert Snap_$i.png nasoEr$i.png -geometry 250x250+1100+800 -composite Snap_$i.png
        convert Snap_$i.png nasovel0$i.png -geometry
        > 250x250+1100+800 -composite Snap_$i.png
        convert Snap_$i.png nasovel2$i.png -geometry
        > 250x250+1400+800 -composite Snap_$i.png
        convert Snap_$i.png nasovel3$i.png -geometry
        > 250x250+1700+800 -composite Snap_$i.png

        convert Snap_$i.png outlet$i.png -geometry
        > 150x150+0+1000 -composite Snap_$i.png
        convert Snap_$i.png refoutlet.png -geometry
        > 150x150+200+1000 -composite Snap_$i.png
```

```
#       convert Snap_$i.png outletEr$i.png -geometry 250x250
+1100+1000 -composite Snap_$i.png
        convert Snap_$i.png outletvel0$i.png -geometry
        > 250x250+1100+1000 -composite Snap_$i.png
        convert Snap_$i.png outletvel2$i.png -geometry
        > 250x250+1400+1000 -composite Snap_$i.png
        convert Snap_$i.png outletvel3$i.png -geometry
        > 250x250+1700+1000 -composite Snap_$i.png

        mv Snap_$i.png Snapshot_$i.png
done

rm *dlt
```

```
#!/bin/bash

reset
set palette maxcolors 2
set palette defined (1 "green", 1.49 "green", 1.51 "red", 2 "red")

#unset grid
set key below horizontal noreverse enhanced autotitle box dashtype solid
set tics out nomirror
set border 3 front linetype black linewidth 4.0 dashtype solid

set grid

#set title 'var'

set xlabel "d (micron)"
set xlabel font "Helvetica,20"
set ylabel "depo"
set ylabel font "Helvetica,20"

set xrange [0:50]
set yrange [0:1]

set tics font ", 18"
set xtics 0,10,40
set ytics 0,0.5,1

set terminal png enhanced
set output 'tmp.png'

set pointsize 3

unset colorbox

plot filename with points pointtype 22 ps 4 palette notitle
```

```python
#!/usr/bin/python

# vtk python script for screen shoting

import glob, string, os, commands, sys
from paraview.simple import *

index = sys.argv[1]

LoadState("/media/milad/ssd0/master_folder/Optimization/postproc/STATE.pvsm")

pm = servermanager.ProxyManager()

s = '/media/milad/Seagate Backup Plus Drive/OPT_CASES'

readerWall = pm.GetProxy('sources', 'smooth.stl')
readerWall.FileNames = s + '/CASE_results.out.' + index + '/constant/triSurface/smooth.stl'
readerWall.FileNameChanged()
readerWall.UpdatePipeline()

#readerObs = pm.GetProxy('sources', 'OBS.stl')
#readerObs.FileNames = s + '/CASE_results.out.' + index + '/constant/triSurface/OBS.stl'
#readerObs.FileNameChanged()
#readerObs.UpdatePipeline()

readerRods = pm.GetProxy('sources', 'RODS.stl')
readerRods.FileNames = s + '/CASE_results.out.' + index + '/constant/triSurface/RODS.stl'
readerRods.FileNameChanged()
readerRods.UpdatePipeline()

view = servermanager.GetRenderView()
#view.Render()
view.StillRender()

#save screenshot
WriteImage("Snap_" + index + ".png")

Delete(readerWall)
#Delete(readerObs)
Delete(readerRods)
Delete(view)
```

Appendix E: Grid Convergence

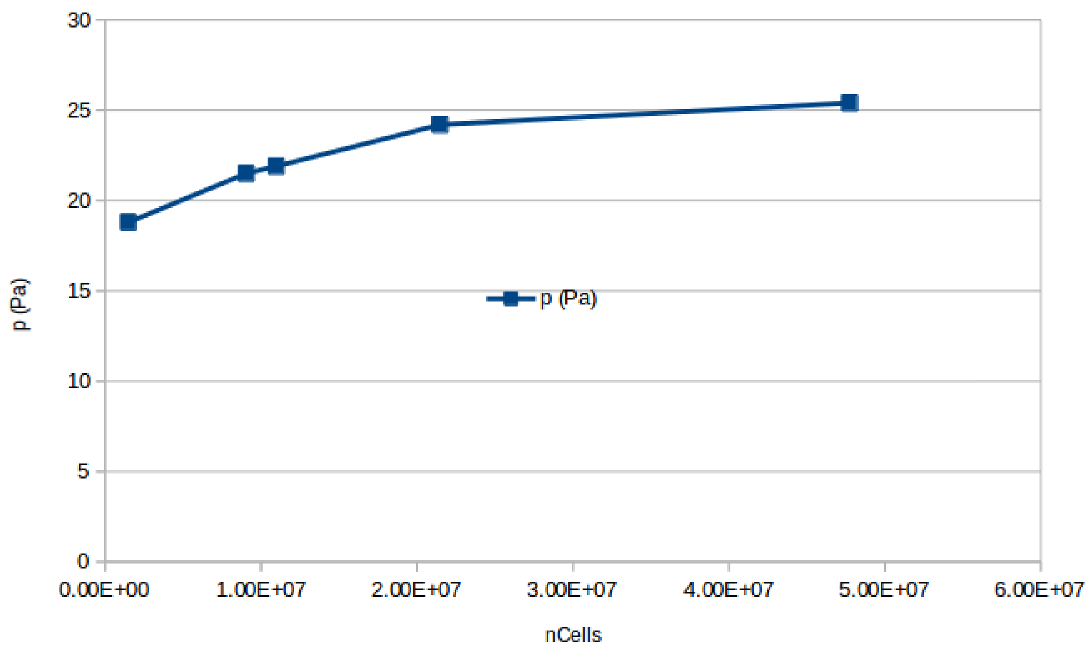


Figure E.1 shows CFD grid convergence study that was performed on subject 1. Due to the computational cost, a few cases were studied. Vertical axis shows the calculated pressure difference between the inlet and the outlet while the horizontal axis shows the number of cells in the grid.